Component Aggregation for PEPA Models: An Approach Based on Approximate Strong Equivalence

Component Aggregation for PEPA Models: An Approach Based on Approximate Strong Equivalence

Dimitrios Milios*, Stephen Gilmore

School of Informatics, University of Edinburgh, UK

Abstract

Approximate aggregation for PEPA components involves the construction of a smaller component that approximates the behaviour of the original one. Such an approximation at the component level can be very efficient and it can also result in a considerable reduction of the state-space for the underlying continuous-time Markov chain. We propose an approximate PEPA component aggregation strategy that relies on an approximate form of strong equivalence. The notion of strong equivalence captures behavioural similarity between components of different size. This quality renders *approximate strong equivalence* appropriate as a criterion to aggregate the state-space of PEPA components. We compare our newly proposed approach with previous work on component aggregation, where only a part of the component behaviour has been used as a criterion for aggregation. Our method requires fewer assumptions regarding the form of the components, and is therefore readily applicable to a larger family of PEPA models.

Keywords: PEPA components, approximate aggregation, strong equivalence

1. Introduction

Informational systems are integral to many aspects of life and the task of performance evaluation remains vitally important as computer infrastructures continue to expand. The rise of the cloud computing industry means that the structure of systems is often extremely complex. Moreover, the massive numbers of people and devices connected to the web, for example in terms of social networks or peer-to-peer networks, not only affect the scale of the system, but also the impact of design decisions.

Markov chains have been extensively used for performance analysis of computer systems that exhibit stochastic behaviour. The PEPA language [1] offers a high-level framework to formally describe complex *Continuous-Time Markov Chains* (CTMCs). A high-level model is defined as a collection of interacting components, and is subsequently translated into a Markov chain, which is amenable to rigorous mathematical analysis. The ability to easily describe complex systems is definitely desirable, however sometimes the resulting state-space is just too large to be handled by traditional analysis techniques.

State-space aggregation can effectively reduce the complexity of large Markov models. If a Markov chain is *lumpable*, then it is possible to construct an aggregated Markov chain featuring transient and steady-state behaviour that is identical to that of the original model w.r.t. the aggregated state-space [2]. In other settings, Markov chain aggregation is characterised as *approximate*, since the behaviour of the original model is approximated by a smaller Markov chain. In a recent work [3], approximate Markov chain aggregation has been defined as a partition optimisation process that relies on the notion of *quasi-lumpability*. This involves the application of a clustering algorithm that minimises an upper bound on a measure that implies quasi-lumpability. For the rest of this paper, we shall refer to this strategy as *QL-based partitioning*. The possibility of a compositional approach to aggregation has also been explored in the same work, a

*Corresponding author

Email addresses: dmilios@staffmail.ed.ac.uk (Dimitrios Milios), stg@staffmail.ed.ac.uk (Stephen Gilmore)

fact that allows the efficient approximation of complex models that consist of a number of components. More specifically, the state-space of PEPA components could be partitioned by the same principle under some assumptions; PEPA components have been treated as CTMCs, where their shared activities have been ignored. This approach made it possible to apply QL-based aggregation to components, rather than the entire state-space. The aggregated underlying CTMC was being produced by utilising the Kronecker representation of PEPA models defined by Hillston & Kloul [4].

One limitation of applying this QL-based approach on PEPA components is that their shared behaviour is completely ignored. As acknowledged in [3], the applicability of such a method is limited to models that are composed of loosely coupled components, in the sense that there is a small number of shared actions. The issue with the shared activities has been that their rates are simply not known at the component level, as they depend on the global state of the system. Shared activity rates are not associated with numerical values that can be used in terms of an optimisation process.

In this work, we propose a partitioning strategy for PEPA components that takes the entire component behaviour into account. This is achieved by formulating estimations for shared activity rates, in order to include these activities in the partitioning process as well. Moreover, we define a partitioning optimisation process that relies on an approximate version of *strong equivalence* [1], which formally describes behavioural equivalence of PEPA components of different size. Most importantly, strong equivalence implies lumpability in the derived Markov chain; therefore we make use of *approximate strong equivalence* (ASE) as a guide to partition the state-space of components. We shall refer to this method as ASE-based partitioning. Finally, we provide a structured operational semantics characterisation of component aggregation. This allows us to produce compositions of approximately aggregated components in a way that eliminates the requirement for an alternative representation based on Kronecker algebra as in [3]. We experimentally evaluate our ASE-based method over a case study inspired by cloud computing, and we present a comparison with the older QL-based approach.

In Section 2, we introduce some concepts used throughout the paper. Section 3 briefly outlines QL-based partitioning of Markov chains originally proposed in [3]. In Section 4, we discuss ASE-based partitioning. The technical details of the clustering algorithms used for component partitioning are discussed in Section 5. In Section 6, we explain how aggregated versions of components can be formed in the case of a partition that only induces approximate strong equivalence. Section 7 presents the operational semantics for PEPA models featuring aggregated components. In Section 8, we present the case-study considered and we discuss the experimental results. In Section 9, we clarify how our work is related to the literature on approximate Markov chain aggregation. Finally, the conclusions of this work are summarised in Section 10.

2. Preliminaries

2.1. The PEPA Language

PEPA models are collections of *components*; the modeller has to specify the components and the way these components interact with each other. The combination of the components can be mapped to a CTMC that can be solved for the transient and the steady-state behaviour of the system. More formally, the grammar for the PEPA language is the following:

$$\begin{array}{lcl} S & ::= & (\alpha, r).S \mid S + S \\ P & ::= & P \underset{\mathcal{L}}{\bowtie} P \mid P/\mathcal{L} \mid S \end{array}$$

where S denotes a sequential component, while P denotes a parallel component which is defined as a composition of sequential components. Below we explain the meaning of the operators and the notation used.

Prefix (.). The prefix operator describes a sequential action that a component may perform. For example, a component $(\alpha, r).P$ carries out an action and subsequently behaves as P. The pair (α, r) is called an *activity*, where α is the action type and r is the *rate* of the activity. The duration of the activity is governed by an exponential distribution with mean $1/r$. The set of activities that a component P is capable of is denoted as $\mathcal{A}ct(P)$.

Choice $(+)$. This operator denotes a choice between two different sequential behaviours. A component $P + Q$ can evolve either to P or Q. Whenever there is a choice between two or more activities, the exponentially distributed transition times imply that there is a race condition between them.

Cooperation $(\underset{\mathcal{L}}{\bowtie})$. This operator denotes a parallel composition of interacting components. A cooperation is defined over a set of actions $\mathcal{L}$, which is called the *cooperation set*. The actions in this set are also called *shared* actions, and they require that the components involved carry out the activity simultaneously. Components may carry out individually any activity whose action type is not in the cooperation set.

Empty Cooperation $(||)$. This operator is a shorthand for $\underset{\emptyset}{\bowtie}$, where the cooperation set is empty.

Aggregation $([N])$. The notation $S[N]$ denotes a collection of identical sequential components that act in parallel with no interaction among them.

Hiding $(/)$. Hiding $P/\mathcal{L}$ renders the actions in the set $\mathcal{L}$ private for the component P. This means that no cooperation can be defined for the actions in $\mathcal{L}$.

Unspecified Rates. The rate of an activity can be *unspecified*, denoted by $\top$, meaning that the activity is *passive*.

Although the rate of passive activities is unspecified, they can also be associated with a probability. This is required if more than one passive activities of the same action type are enabled. Then an unspecified activity rate $\top$ may be assigned a weight $w \in \mathbb{N}$ which represents the relative probability of that particular activity. The absence of a weight simply implies that $w = 1$. For example, the component $P \stackrel{def}{=} (\alpha_1, w_1 \times \top).P_1 + (\alpha_2, w_2 \times \top).P_2$, will perform α_1 with probability $w_1/(w_1 + w_2)$ or α_2 with probability $w_2/(w_1 + w_2)$. The time of exiting state P remains unspecified.

2.1.1. The Derivation Graph

The Markovian interpretation of a PEPA model relies on the semantics of the PEPA language [1] summarised in Fig. 1, which are defined in the style of Plotkin's *structured operational semantics* [5]. For a component P, the PEPA semantics induces the set of states reachable from P, which is called the *derivative set*, denoted as $ds(P)$. The derivative set along with the activities involved define the *derivation graph*. If the derivation graph contains no unspecified rates, then we can construct a CTMC $(ds(P), \boldsymbol{Q}, \boldsymbol{\pi}_0)$, where $\boldsymbol{\pi}_0$ is some initial distribution over $ds(P)$, while $\boldsymbol{Q}$ is a generator matrix whose entries capture the transition rates of the derivation graph.

In the first level of syntax, the modeller defines one or more sequential components. The second level of syntax is the *system equation*, which specifies which components participate in the system and what are the interactions among them. The local states of the components formulate the global state of the system. Any change in the local states will also have an effect on the system state. In the case of individually performed actions, the transitions are associated with an exponential distribution as described earlier. For a shared action however, the components involved have to perform this action at the same time. The duration of a synchronised transition will follow an exponential distribution that is determined by the activity with the smallest rate. In order to see how, we have to explain the notion of *apparent rate* introduced in [1]:

Definition 1 (Apparent Rate)**.** *The apparent rate of an action α in a component P, which is denoted as $r_\alpha(P)$, is the sum of all rates of all activities of type α in $\mathcal{A}ct(P)$.*

As with activity rates, an apparent rate can be either a positive real number, or unspecified $\top$ with weight equal to the sum of the weights of the activities included. Note however that the apparent rate $r_\alpha(P)$ can only be defined if the activities of type α for P are either all active or all passive, since the addition of an active rate $r \in R^+$ with an unspecified rate $\top$ is not defined as an operation in [1]. Components for which any action of type α involves both active and passive activities are considered invalid. This requirement has some interesting implications in an approximate aggregation setting, which we explore later in Section 6.3.

Prefix

$$\frac{}{(\alpha, r).E \xrightarrow{(\alpha,r)} E}$$

Cooperation

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E' \underset{L}{\bowtie} F} \ (\alpha \notin L) \qquad \frac{F \xrightarrow{(\alpha,r)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E \underset{L}{\bowtie} F'} \ (\alpha \notin L)$$

$$\frac{E \xrightarrow{(\alpha,r_1)} E' \quad F \xrightarrow{(\alpha,r_2)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,R)} E' \underset{L}{\bowtie} F'} \ (\alpha \in L) \qquad \text{where} \quad R = \frac{r_1}{r_\alpha(E)} \frac{r_2}{r_\alpha(F)} \min(r_\alpha(E), r_\alpha(F))$$

Choice

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E + F \xrightarrow{(\alpha,r)} E'} \qquad \frac{F \xrightarrow{(\alpha,r)} F'}{E + F \xrightarrow{(\alpha,r)} F'}$$

Hiding

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\alpha,r)} E'/L} \ (\alpha \notin L) \qquad \frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\tau,r)} E'/L} \ (\alpha \in L)$$

Constant

$$\frac{E \xrightarrow{(\alpha,r)} E'}{A \xrightarrow{(\alpha,r)} E'} \ (A \stackrel{def}{=} E)$$

Figure 1: Structured operational semantics for PEPA

The concept of apparent rate refers to the rate of any currently enabled activity of a given action type. The rate of a synchronised activity will be the minimum of the two apparent rates involved, weighted by the individual activity probabilities. More specifically, the rate of the synchronised activities $P \stackrel{def}{=} (\alpha, r_1).P'$ and $Q \stackrel{def}{=} (\alpha, r_2).Q'$ will be

$$r = \frac{r_1}{r_\alpha(P)} \frac{r_2}{r_\alpha(Q)} \min(r_\alpha(P), r_\alpha(Q)) \tag{1}$$

In the special case where there is one or more passive actions, the unspecified rate $\top$ is evaluated in terms of the minimum function as follows:

$$\min(w_1 \times \top, r) = \begin{cases} r, & r \in \mathbb{R}^+ \\ \min(w_1, w_2) \times \top, & r = w_2 \times \top \end{cases} \tag{2}$$

where $w_1, w_2 \in \mathbb{N}$.

2.1.2. PEPA Components as Labelled Transition Systems

In the PEPA language, a component name C is bound to a process definition. The derivative set $ds(C)$ along with the activities involved define a labelled transition system. In terms of the current paper, a component C will actually refer to the labelled transition system with state-space $ds(C)$, and transitions which are specified by the semantics of Fig. 1. In other words, our use of the term "component" involves the set of process definitions for which the semantics induce a directed graph with one connected component.

According to standard PEPA notation [1], the set of actions of which a component C is capable is denoted by $\mathcal{A}(C)$. Since we are interested in the labelled transition system induced by C, we shall make use of the *complete set of actions* of a component C, which is defined as follows:

$$\vec{\mathcal{A}}(C) = \bigcup_{C' \in ds(C)} \mathcal{A}(C') \tag{3}$$

Assuming an ordering of states in the derivative set $ds(C)$, we shall describe the behaviour of components in terms of matrices, whose entries contain the rates of each individual activity. Given the complete set of actions of C, we have an *action rate matrix* $\boldsymbol{R}_a$ for every action $a \in \vec{\mathcal{A}}(C)$. The entries of each $\boldsymbol{R}_a$ matrix are determined by the language semantics:

- An entry $R_{a,ij} > 0$ denotes the rate of the activities of type a from the i-th state to the j-th state of C.
- A value 0 for $R_{a,ij}$ means that there is no activity of type a from the i-th to the j-th state of C.
- An unspecified rate $w \times \top$ for $R_{a,ij}$ implies that the activities of type a from the i-th to the j-th state of C are passive, having weight $w \in \mathbb{N}$.

2.2. Notions of Equivalence

In order to aggregate a Markov chain with N states, its state-space has to be partitioned into K *classes*, where ideally $K \ll N$. Given a state-space S, we shall say that $\Delta = \{A_1, \ldots, A_K\}$ is a *partition* on S with K classes, where $A_1, \ldots, A_K$ are mutually disjoint subsets of S. The states that belong to the same class have to be equivalent in some sense. In this section, we review some notions of exact and approximate state equivalence in Markov chains and PEPA components. A large part of the discussion in this paper relies on the concepts that follow.

2.2.1. Lumpability

In terms of Markov chains, equivalence is formally described by the notion of *lumpability* [6]. Given a partition of the state-space, lumpability implies that states that belong to the same class have identical transition probabilities to each of the partitions. This concept is formally described by the following definition:

Definition 2 (Lumpability)**.** *A Markov chain with probability matrix $\boldsymbol{P}$ is lumpable w.r.t. a partition $\Delta = \{A_1, \ldots, A_K\}$, if for any two classes $A_k, A_l \in \Delta$, and for any two states $i, j \in A_k$:*

$$\sum_{m \in A_l} P_{im} = \sum_{m \in A_l} P_{jm} \tag{4}$$

As can be seen in [2], given a lumpable Markov chain we can obtain a *lumped* model which is also a Markov chain. Given a stochastic matrix $\boldsymbol{P} \in \mathbb{R}^{N \times N}$ that is lumpable to $\Delta = \{A_1, \ldots, A_K\}$, we define the corresponding lumped matrix $\tilde{\boldsymbol{P}} \in \mathbb{R}^{K \times K}$ with entries:

$$\tilde{P}_{kl} = \sum_{m \in A_l} P_{im}, \quad \forall i \in A_k \tag{5}$$

The lumped matrix $\tilde{\boldsymbol{P}}$ has transient and steady-state behaviour that is identical to the behaviour of $\boldsymbol{P}$ w.r.t. the aggregated state-space.

The concept of lumpability can be easily extended to CTMCs by considering the infinitesimal generator matrix $\boldsymbol{Q}$ instead of the probability matrix $\boldsymbol{P}$. It is also well known that if a CTMC is lumpable to some partition Δ, the discrete-time Markov chain obtained via uniformisation is also lumpable to Δ. State-space aggregation techniques that rely on lumpability typically exploit the structure of some high-level description of the model. For example in [7], a lumpable partition is obtained by identifying isomorphic components of a PEPA model.

2.2.2. Quasi-Lumpability

In the general case, a non-trivial lumpable partition might not exist. *Quasi-lumpability*, which was introduced in [8], captures approximate behaviour for Markov models. In order to describe states that exhibit approximately the same rather than identical behaviour, we have to relax the conditions in (4).

Definition 3 (Quasi-Lumpability)**.** *A Markov Chain with probability matrix* $\boldsymbol{P}$ *will be quasi-lumpable w.r.t. a partition* $\Delta = \{A_1, \ldots, A_K\}$ *and a bound* ϵ*, if for any two classes* $A_k, A_l \in \Delta$*, and for any two states* $i, j \in A_k$*:*

$$\left| \sum_{m \in A_l} P_{im} - \sum_{m \in A_l} P_{jm} \right| \leq \epsilon, \quad \epsilon \geq 0 \tag{6}$$

The term *near-lumpability* has been used to describe the same notion in [2], however we shall use the term "quasi-lumpability" for the rest of this paper. Most of the research in the field so far aims at computing bounds for the state probabilities of quasi-lumpable Markov chains, assuming some partition of the state-space [8, 9, 10]. The computation of bounds of compositions of Markov chains has also been investigated in the context of Markov reward models [11] and PEPA [12].

2.2.3. Strong Equivalence

In the original work on PEPA [1], several equivalences for PEPA components have been introduced, including isomorphism, weak isomorphism, strong bisimilarity and strong equivalence. Isomorphism is not really suitable for our purposes here, as the components involved are required to have state-spaces of the same size. Weak isomorphism and strong bisimilarity allow components of different size, however they do not induce lumpability for the underlying CTMCs.

Strong equivalence combines the two desirable features needed to define approximate component equivalence in the style of quasi-lumpability. It has been shown in [1] that if two components C and $\tilde{C}$ are strongly equivalent then their underlying CTMCs will be lumpably equivalent. Moreover, the compositions $C \underset{\mathcal{L}}{\bowtie} C'$ and $\tilde{C} \underset{\mathcal{L}}{\bowtie} C'$ will be also strongly equivalent for any C'.

In the context of state-space aggregation, a component $\tilde{C}$, which is strongly equivalent to C, results from aggregation given some partition on the state-space of C. Strong equivalence can then be defined in terms of the action rate matrices and a partition Δ on the state-space of C.

Definition 4 (Strong Equivalence)**.** *Let* C *be a PEPA component with transition rate matrices* $\boldsymbol{R}_a$*,* $\forall a \in \vec{\mathcal{A}}(C)$*. A partition* $\Delta = \{A_1, \ldots, A_K\}$ *on* $ds(C)$ *induces a strongly equivalent aggregated component, if for any two classes* $A_k, A_l \in \Delta$*, and for any two states* $i, j \in A_k$*:*

$$\sum_{m \in A_l} R_{a,im} = \sum_{m \in A_l} R_{a,jm}, \quad \forall a \in \vec{\mathcal{A}}(C) \tag{7}$$

If we relax the conditions of strong equivalence in a way similar to quasi-lumpability, we obtain the following approximate notion of equivalence for PEPA components:

Definition 5 (Approximate Strong Equivalence)**.** *Let* C *be a PEPA component with transition rate matrices* $\boldsymbol{R}_a$*,* $\forall a \in \vec{\mathcal{A}}(C)$*. A partition* $\Delta = \{A_1, \ldots, A_K\}$ *on* $ds(C)$ *induces an approximately strongly equivalent aggregated component with respect to a bound* ϵ*, if for any two classes* $A_k, A_l \in \Delta$*, and for any two states* $i, j \in A_k$*:*

$$\left| \sum_{m \in A_l} R_{a,im} - \sum_{m \in A_l} R_{a,jm} \right| \leq \epsilon, \quad \epsilon \geq 0, \quad \forall a \in \vec{\mathcal{A}}(C) \tag{8}$$

3. Partitioning Based on Quasi-Lumpability

The QL-based partitioning approach presented in [3] has been originally defined in terms of Markov chains. Given certain assumptions, PEPA components could be partitioned under the same principle, which

makes it possible for approximate aggregation to be applied in a compositional setting. In this section, we outline QL-based partitioning of Markov chains [3], and we discuss the conditions required so that this strategy is effectively applied for PEPA components.

3.1. Markov Chain Partitioning

As we have seen in Definition 3, quasi-lumpability resulted from relaxing the conditions imposed for lumpable Markov chains. In quasi-lumpable models, the class-to-class transition probabilities have to be approximately the same for states that belong to the same class. More formally, given a Markov chain with probability matrix $\boldsymbol{P}$ that is quasi-lumpable w.r.t. $\Delta = \{A_1, \ldots, A_K\}$, then for any two classes $A_k, A_l \in \Delta$ and for any two states $i, j \in A_k$ we have:

$$\left| \sum_{m \in A_l} P_{im} - \sum_{m \in A_l} P_{jm} \right| \leq \epsilon, \quad \epsilon \geq 0$$

The quantity ϵ above represents the maximum difference between states in the same class. A partitioning strategy that relies on the concept of quasi-lumpability should minimise ϵ for every combination of states. Under this perspective, we can use the following pseudo-metric to capture a dissimilarity measure between states:

$$E_{i,j} = \sum_{l=1}^{K} \left| \sum_{m \in A_l} P_{im} - P_{jm} \right| \tag{9}$$

where i, j are states that belong to the same class. This pseudo-metric will be equal to zero, only if the corresponding Markov chain is lumpable with respect to the partition Δ.

In theory, a partition can be considered as optimal with respect to the notion of quasi-lumpability, if it minimises all of the $E_{i,j}$ quantities. However, in (9) the dissimilarity between a pair of states is not constant, as it depends on the way that the states are distributed across the classes. Therefore, it is not straightforward to design an algorithm that directly minimises $E_{i,j}$ with respect to the partition. The solution proposed in [3] involves minimising $d(i, j) \geq E_{i,j}$ instead, which is the *Manhattan distance* in the $\mathbb{R}^N$ space spanned by the transition probabilities to the N states of a Markov chain:

$$d(i, j) = \sum_{n=1}^{N} |P_{in} - P_{jn}| \tag{10}$$

In order to obtain a partitioning of the state-space that minimises the Manhattan distance for states in the same class, we have to apply a clustering algorithm. Minimising $d(i, j)$ also means small values for $E_{i,j}$, and subsequently for the ϵ quantity in the definition of quasi-lumpability. This result also applies for CTMCs by considering the embedded discrete-time Markov chain obtained via *uniformisation* [13]. The technical details of the clustering algorithm of our choice are discussed in Section 5.

3.2. QL-based Partitioning for PEPA Components

Let us consider a PEPA component C as a labelled transition system with action rate matrices $\{\boldsymbol{R}_a, \forall a \in \vec{\mathcal{A}}(C)\}$. If we ignore the action labels, we can simply calculate the sum $\boldsymbol{R} = \sum_{a \in \vec{\mathcal{A}}(C)} \boldsymbol{R}_a$, which is the transition rate matrix of the underlying CTMC. A partition on the state-space of C can be obtained by using the QL-based partitioning method discussed in the previous section. However, the $\boldsymbol{R}$ matrix cannot be calculated at all times, as some of the activities are shared, meaning that the corresponding rates depend on other components. At this point, it is useful to discriminate between the *shared* and the *individual behaviour* of a PEPA component by introducing the following definition:

Definition 6. *Let C be a PEPA component whose individual activities have action type τ. We shall say that the individual behaviour of C is characterised by the* individual rate matrix $\boldsymbol{R}_\tau$*, while the shared behaviour of C is characterised by the rate matrices: $\boldsymbol{R}_a$, $\forall a \neq \tau \in \vec{\mathcal{A}}(C)$.*

Without loss of generality, we can only consider PEPA components whose individual activities have a single action type, namely τ. A useful observation regarding Definition 6 is that we can summarise the individual behaviour of any component in a single rate matrix. For the shared actions however, there is no legitimate way to produce a meaningful summary in a single matrix. Recall that the rates of these actions are not fully determined in isolation, as they depend on the global state. More specifically, the PEPA semantics impose that the rate of a shared activity is the minimum of the apparent rates of the components involved. This minimum also depends on the state of the other cooperating component, so it cannot be determined *a priori* i.e. without deriving the global state-space.

According to the PEPA semantics, the individual activities of a component C induce a CTMC, whose generator matrix can be directly derived by appropriately modifying the diagonal entries of the individual rate matrix $\boldsymbol{R}_\tau$. Note that this CTMC does not accurately capture the behaviour of C, as the shared actions have been ignored. However, if the set of shared actions is relatively small, we can expect that $\boldsymbol{R}_\tau$ will be much more dense than the total of $\boldsymbol{R}_a$, $\forall a \neq \tau \in \vec{\mathcal{A}}(C)$. If this condition holds and $\boldsymbol{R}_\tau$ captures a significant part of the component behaviour, it should be reasonable to apply an approximate aggregation algorithm to $\boldsymbol{R}_\tau$, in order to obtain a nearly optimal partition Δ with respect to the individual behaviour of C. These claims have been experimentally evaluated in [3], where it was found that such a partitioning strategy could adequately capture the behaviour of a class of models that involve both low and high population components. Further evaluation and comparison with the more sophisticated ASE-based approach follow in Section 8.

4. Partitioning Based on Approximate Strong Equivalence

In this section, we discuss how approximate strong equivalence can be used as a criterion to partition the state-space of PEPA components. Moreover, we explain how we can produce estimations about the rates of shared activities, which are generally not known at the component level. In this way, shared activities can be associated with certain values that can be used in terms of a numerical algorithm.

4.1. A Pseudo-metric related to Strong Equivalence

In terms of PEPA components, we need a measure to capture behavioural similarity between states. It will be convenient for the arguments that follow to introduce the following definition:

Definition 7 (Uniformisation of PEPA Components)**.** *Let C be a PEPA component with transition rate matrices $\boldsymbol{R}_a$, $\forall a \in \vec{\mathcal{A}}(C)$. We define a* uniformisation constant *γ for C as follows:*

$$\gamma \geq \max_{C' \in ds(C)} \sum_{a \in \vec{\mathcal{A}}(C)} r_a(C') \tag{11}$$

We construct a component $\mathring{C}$ with action types $\vec{\mathcal{A}}(\mathring{C}) = \vec{\mathcal{A}}(C)$, derivation set $ds(\mathring{C}) \cong ds(C)$, apparent rates:

$$r_a(\mathring{C}') = \gamma, \quad \forall \mathring{C}' \in ds(\mathring{C}), \quad \forall a \in \vec{\mathcal{A}}(\mathring{C}) \tag{12}$$

and transition probability tensor $\mathring{\boldsymbol{P}} \in \mathbb{R}^{|ds(\mathring{C})| \times |ds(\mathring{C})| \times |\vec{\mathcal{A}}(\mathring{C})|}$ with entries:

$$\mathring{P}_{ija} = \begin{cases} \dfrac{1}{\gamma} R_{a,ij}, & i \neq j, \quad \forall a \in \vec{\mathcal{A}}(\mathring{C}) \\ \dfrac{1}{\gamma} R_{a,ii}, & i = j, \quad a \neq \tau \\ 1 - \dfrac{\sum_{a \in \vec{\mathcal{A}}} \sum_j R_{a,ij}}{\gamma}, & i = j, \quad a = \tau \end{cases} \tag{13}$$

We shall say that $\mathring{C}$ is the uniformisation of C.

Lemma 1. *A component C is strongly equivalent to its uniformisation $\mathring{C}$.*

Proof. This can be trivially shown if we multiply $\mathring{\boldsymbol{P}}$ with the uniformisation constant γ. □

Similarly to the concept of uniformisation of CTMCs, a uniformised PEPA component is associated with a jump process that describes the next-step transition probabilities, and a uniformisation constant that expresses the rate of those jumps. The only difference is that we must have a different dimension for each shared action, as we do not want their effects to be counterbalanced. In Definition 7, the entire component behaviour has been summarised by a single structure; that is the tensor $\mathring{\boldsymbol{P}}$, whose entries describe the absolute activity probabilities, assuming a total rate γ.

We can derive a pseudo-metric that captures the similarity measure between states of a PEPA component. This measure will have to rely on the notion of strong equivalence, which can be conveniently expressed in terms of the probability tensor of a uniformised PEPA component $\mathring{C}$. Considering a partition $\Delta = \{A_1, \ldots, A_K\}$ on the state-space of $\mathring{C}$, we define the following quantity for any two states i, j that belong to the same class:

$$\mathring{E}_{i,j} = \sum_{a \in \vec{\mathcal{A}}(\mathring{C})} \sum_{l=1}^{K} \left| \sum_{m \in A_l} \mathring{P}_{ima} - \mathring{P}_{jma} \right| \tag{14}$$

In the equation above, $\mathring{E}_{i,j}$ is characterised as a pseudo-metric, as we may have $\mathring{E}_{i,j} = 0$ for $i \neq j$. What makes this measure appropriate to determine similarity for states of PEPA components is the fact that each activity contributes to the measure at a different dimension, depending on its type. $\mathring{E}_{i,j}$ will be equal to zero iff Δ induces a strongly equivalent aggregated component for $\mathring{C}$. In a different case, Δ can be characterised as optimal, if it minimises the pairwise distances $\mathring{E}_{i,j}$ for any two states in the same class.

Similarly to what we have done for the quasi-lumpability measure, we can define an upper bound for $\mathring{E}_{i,j}$ as follows:

$$\begin{aligned} \mathring{d}(i,j) &= \sum_{a \in \vec{\mathcal{A}}(\mathring{C})} \sum_{l=1}^{K} \sum_{m \in A_l} \left| \mathring{P}_{ina} - \mathring{P}_{jna} \right| \\ &= \sum_{a \in \vec{\mathcal{A}}(\mathring{C})} \sum_{n=1}^{N} \left| \mathring{P}_{ina} - \mathring{P}_{jna} \right| \end{aligned} \tag{15}$$

where $N = |ds(\mathring{C})|$. It is easy to show that $\mathring{d}(i,j) \geq \mathring{E}_{i,j}$, where $\mathring{d}(i,j)$ is the Manhattan distance between two rows of the probability tensor. This distance is readily applicable for use in terms of a clustering algorithm; the output of such an algorithm will be a partition on the state-space of $\mathring{C}$ that minimises the pairwise Manhattan distances for states in the same class. Since $\mathring{d}(i,j)$ is also an upper bound for $\mathring{E}_{i,j}$, then it reasonable to expect that the same partition will induce small values for the $\mathring{E}_{i,j}$ measure.

We note that approximate equivalence could have been alternatively defined in terms of *lumpable bisimulation*, which is a characterisation of *contextual lumpability* [14]. It can be easily seen that the Manhattan distance is also an upper bound for lumpable bisimulation, therefore this alternative definition should have no effect on the partitioning strategy that we propose, as it relies on the Manhattan distance only.

4.2. Estimating the Rates of Shared Activities

If a is an individual action, then the entries of the corresponding rate matrix $\boldsymbol{R}_a$ are all known. In the case where a is a shared action however, the activity rates may be either unspecified or simply determined by other slower components. In order to examine the possibilities for shared activities, we shall decompose $\boldsymbol{R}_a$ into apparent rates and activity probabilities.

Definition 8 (Apparent Rate Vector)**.** *Let C be a PEPA component whose state-space is indexed by $ds(C)$. Let $\boldsymbol{r}_a$ be a vector, also indexed by $ds(C)$, which contains the apparent rates of the states of C for action a. We define $\boldsymbol{r}_a$ to be the* apparent rate vector *of C for action a.*

Using Definition 8, we can decompose any rate matrix $\boldsymbol{R}_a$ as a product of two matrices:

$$\boldsymbol{R}_a = \boldsymbol{D}_a \boldsymbol{P}_a \tag{16}$$

where $\boldsymbol{D}_a$ is a diagonal matrix with entries $\boldsymbol{r}_a$, and $\boldsymbol{P}_a$ is the probability matrix for the activities of type a. For a shared action a, the activity probabilities in $\boldsymbol{P}_a$ are completely characterised by the component specification. However, the apparent rates in $\boldsymbol{r}_a$ can be either unspecified or determined by slower components. In both cases, there is uncertainty about the entries of $\boldsymbol{R}_a$, hence it cannot be used in terms of a partitioning strategy in its current form.

In practice, a particular shared activity may take values within a range of values, depending on the state of any cooperating components. In this section, we discuss a strategy to produce estimations for the apparent rates of shared actions. These estimations constitute the maximum possible capacity of a given action for a particular component state. In this way, we capture the maximum possible effect that a shared activity may have in the uniformised probability tensor.

Let us assume components P and Q that cooperate on some action a. The apparent rates of P for action a are summarised in the vector $\boldsymbol{r}_{a,P}$. Similarly, $\boldsymbol{r}_{a,Q}$ is the apparent rate vector for Q. Note that the components, and subsequently the corresponding apparent rate vectors, do not have to be of the same size. According to the PEPA semantics, the apparent rate for a synchronised action will be the minimum of the apparent rates for the components involved. For a particular state i in P we have the following vector of possible values for the apparent rate:

$$\min(r_{i,a,P}, \boldsymbol{r}_{a,Q}) = \underbrace{\begin{bmatrix} \min(r_{i,a,P}, r_{1,a,Q}) \\ \min(r_{i,a,P}, r_{2,a,Q}) \\ \cdots \end{bmatrix}}_{|ds(Q)| \text{ entries}} \tag{17}$$

We see that the apparent rate $r_{i,a,P}$ will be determined solely by P if $r_{i,a,P}$ is smaller than all of the apparent rates in Q. In other cases, there is uncertainty about the apparent rate. Our strategy is to consider the largest possible apparent rate; that will correspond to the maximum value in the apparent rate collection. Therefore, for the components P and Q we define the *estimated* apparent rate vectors $\boldsymbol{r}'_{a,P}$ and $\boldsymbol{r}'_{a,Q}$ correspondingly with entries:

$$\begin{aligned} r'_{i,a,P} &= \max_j(\min(r_{i,a,P}, r_{j,a,Q})) \\ r'_{i,a,Q} &= \max_j(\min(r_{i,a,Q}, r_{j,a,P})) \end{aligned} \tag{18}$$

We can use (18) to recursively estimate the maximum possible rates for any term of the abstract syntax tree that corresponds to a PEPA system equation. For each sequential component, we construct the apparent rate vector. Starting from the leaves of the system equation, we resolve the apparent rate vectors of the lower level, and we use the resulting rate vectors in the upper level. Algorithm 1 formally describes this recursive process of constructing the estimated apparent rate vectors based on (18). We shall say that a component C is an element of a component M, that is $C \in M$, if C is part of the composition that forms the parallel component M. We see that for any pair of components $C, C' \in M$ that cooperate, either directly or indirectly, we calculate estimated apparent rates. Any intermediate results are stored and used in later calculations, so it is possible to produce rate estimations in the case of multi-way synchronisation. Hidden activities should be treated as individual activities, regardless of their presence in a cooperation set. In the practical application of the algorithm, parallel compositions of N identical sequential components that act with no interaction among them, i.e. $C[N]$, are considered as single sequential components.

We acknowledge that Algorithm 1 may not calculate the true maximum possible apparent rates if there is an *implicit choice* in the model. For example, consider a system equation of the form $P \underset{\{a\}}{\bowtie} (Q||R)$, where $a \in \vec{\mathcal{A}}(P) \cap \vec{\mathcal{A}}(Q) \cap \vec{\mathcal{A}}(R)$. According to (18), the total capacity of P for a might be limited because of either Q or R. In that case, since this is considered as an intermediate result, the total capacity of R for a

Algorithm 1 Recursive Expectation for Apparent Rate Vectors

Require: M is a PEPA model definition
Ensure: The rate vectors $\{\boldsymbol{r}_{a,C} : a \in \vec{\mathcal{A}}(C), C \in M\}$ contain estimations for the apparent rates

if M is a sequential PEPA component **then**
  return $\{\boldsymbol{r}_{a,M} : a \in \vec{\mathcal{A}}(M)\}$
else
  $P \underset{\mathcal{L}}{\bowtie} Q \leftarrow M$
  Apply this algorithm recursively to obtain the set: $\{\boldsymbol{r}_{a,C} : a \in \vec{\mathcal{A}}(C), C \in P\}$
  Apply this algorithm recursively to obtain the set: $\{\boldsymbol{r}_{a,C} : a \in \vec{\mathcal{A}}(C), C \in Q\}$
  for all $C \in P$ **do**
    for all $C' \in Q$ **do**
      for all $a \in \mathcal{L} \cap \vec{\mathcal{A}}(C) \cap \vec{\mathcal{A}}(C')$ **do**
        Construct the apparent rate vector $\boldsymbol{r}'_{a,C}$ with entries: $r'_{i,a,C} = \max_j(\min(r_{i,a,C}, r_{j,a,C'}))$
        Construct the apparent rate vector $\boldsymbol{r}'_{a,C'}$ with entries: $r'_{i,a,C'} = \max_j(\min(r_{i,a,C'}, r_{j,a,C}))$
        $\boldsymbol{r}_{a,C} \leftarrow \boldsymbol{r}'_{a,C}$
        $\boldsymbol{r}_{a,C'} \leftarrow \boldsymbol{r}'_{a,C'}$
      end for
    end for
  end for
  return $\{\boldsymbol{r}_{a,C} : a \in \vec{\mathcal{A}}(C), C \in P\} \cup \{\boldsymbol{r}_{a,C} : a \in \vec{\mathcal{A}}(C), C \in Q\}$
end if

will also be limited because of Q, while it should not. Nevertheless, Algorithm 1 will still give estimations regarding the apparent rates that can be used in terms of component aggregation. The modeller has to bear in mind that the estimated maximum rates calculated may be underestimations of the actual maximum rates. However, the existence of an implicit choice can be easily detected at syntax level, and therefore it can be avoided if the modeller sees fit.

As a remark in the computational complexity of Algorithm 1, every component C has to be pair with every C' that is on the other side of a cooperation operator. If we have n sequential components in a model, a rough estimate of the worst-case complexity will be of $O(n^2 \times n_a)$, where by n_a we denote the total number of actions. This does not present an issue from a practical point of view however, since the algorithm is meant to be applied at the system equation level. Even the most complex models do not normally involve more than a few tens of compositions.

5. The Clustering Algorithm

The approximate aggregation approaches discussed in the previous two sections rely on partitioning the state-space of a component C according to an appropriately defined Manhattan distance metric. The clustering algorithm of our choice is the one proposed by Ng et al. in [15], which is a *spectral clustering* approach [16]. Using an appropriate notion of similarity, a dataset is associated with a weighted undirected graph that is called the *similarity graph*. This similarity graph is summarised by the *affinity matrix* $\boldsymbol{S}$, whose entries $S_{ij} = S_{ji} \geq 0$ denote the pairwise similarities between the i-th and the j-th instances. The data is partitioned depending on the eigenvectors of the *Laplacian* matrix, rather than on the local proximities of data-points. In [15] in particular, the following symmetric version of the Laplacian matrix is used:

$$\boldsymbol{L}_{sym} = \boldsymbol{D}^{-1/2}\boldsymbol{S}\boldsymbol{D}^{-1/2} \tag{19}$$

where $\boldsymbol{D}$ is a diagonal matrix whose entries are the row sums of $\boldsymbol{S}$.

Algorithm 2 describes our adaptation of Ng et al. [15] to partition the state-space of PEPA components. The algorithm has as input a PEPA component C, and partitions its state-space using either its individual

rate matrix $\boldsymbol{R}_\tau$ (QL-based method), or a probability tensor obtained via component uniformisation (ASE-based method). On the technical part, the algorithm selects the K eigenvectors that correspond to the K largest eigenvalues of $\boldsymbol{L}_{sym}$. Then the data is mapped to the rows of the $\boldsymbol{X} \in \mathbb{R}^{N\times K}$ matrix formed by stacking these eigenvectors as columns. The clusters of data are well separated in this $\mathbb{R}^K$ space, therefore the premise of spectral clustering methods is that they produce a globally optimal clustering.

Algorithm 2 Spectral Clustering of PEPA Components

Require: A PEPA component C with rate matrices $\boldsymbol{R}_\tau$ and $\{R_a, \forall a \neq \tau \in \vec{\mathcal{A}}(C)\}$

1: **QL-method** Calculate $P \in \mathbb{R}^{N\times N}$ as the uniformisation of $\boldsymbol{R}_\tau$

ASE-method

a) Apply Algorithm 1 to estimate the maximum possible apparent rates for the components in the model

b) Calculate $P \in \mathbb{R}^{|ds(\mathring{C})|\times|ds(\mathring{C})|\times|\vec{\mathcal{A}}(\mathring{C})|}$ such that $\mathring{C}$ is the uniformisation of C

2: Compute the affinity matrix $\boldsymbol{S} \in \mathbb{R}^{N\times N}$ with entries:

QL-method $S_{ij} = \exp\left(-\frac{d(i,j)}{2\sigma^2}\right)$

ASE-method $S_{ij} = \exp\left(-\frac{\mathring{d}(i,j)}{2\sigma^2}\right)$

3: Construct the diagonal matrix $\boldsymbol{D}$ with entries $D_{ii} = \sum_{j=1}^{N} S_{ij}$

4: Construct the symmetric Laplacian: $\boldsymbol{L}_{sym} = \boldsymbol{D}^{-1/2}\boldsymbol{S}\boldsymbol{D}^{-1/2}$

5: Find the eigenvectors that correspond to the K largest eigenvalues of $\boldsymbol{L}_{sym}$ and form the matrix:

$$\boldsymbol{X} = [\boldsymbol{x_1}\boldsymbol{x_2}\ldots\boldsymbol{x_K}] \in \mathbb{R}^{N\times K}$$

6: Normalise each row of $\boldsymbol{X}$ so as it has unit length:

$$Y_{ij} = \frac{X_{ij}}{\sqrt{\sum_{j=1}^{K} X_{ij}^2}}$$

7: Given that each row of $\boldsymbol{Y}$ is a data-point in $\mathbb{R}^K$, perform a K-means clustering

Step 2 of Algorithm 2 involves the computation of a similarity measure between two states given their distance. As we can see, the definition of similarity depends on the parameter σ^2, which controls how quickly the affinity S_{ij} degrades. In fact, σ^2 affects how wide a cluster can be, i.e. large values for σ^2 favour wide clusters. As noted in [16], spectral clustering algorithms do not make strong assumptions on the form of the clusters. A too large value for σ^2 could lead to cluster shapes that are not meaningful in terms of quasi-lumpability, where we would only want to minimise the Manhattan distance between each pair of states in the same cluster. Therefore, we need a value for σ^2 that imposes "tight" clusters. Since we use stochastic matrices only (the generator matrices of CTMCs are uniformised), we know that the maximum Manhattan distance between any two rows is 2. In Fig. 2 we can see the effect of different σ^2 values on the similarity function. An appropriate value to enforce tight clusters is 0.1, which has been used in the experiments later in this work.

The final step of Algorithm 2 clusters the data according to their affinities when mapped onto the space spanned by the K largest eigenvectors of the Laplacian $\boldsymbol{L}_{sym}$. This process of clustering is described in a rather abstract way as the output of a *K-means* clustering approach. This step requires further explanation, as K-means cannot guarantee a globally optimal partition. In most implementations, it starts from a random initial solution and performs a number of iterations until it converges to a local optimum [17]. In practice,

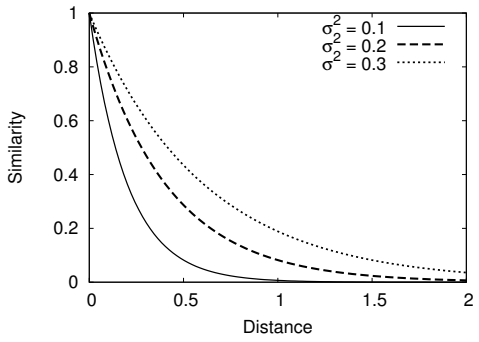


Figure 2: The similarity function $\exp\left(-\frac{x}{2\sigma^2}\right)$ for different values of σ^2

multiple runs are required to obtain a globally optimal solution.

The cost of performing K-means has been significantly reduced in terms of the Ng et al. method [15] using an appropriate initialisation. Note that a normalised version of the eigenvectors is used, which are stored as columns in the $\boldsymbol{Y}$ matrix. In the ideal case, that is when the Markov chain is completely decomposable, the points $Y_i \in \mathbb{R}^K$ have the form $(0, \ldots, 0, 1, 0, \ldots, 0)$ where the position of the "1" indicates the connected component this point belongs to. Notice that instances that belong to different clusters are orthogonal to each other. Perturbation analysis that was performed in [15] shows that this property is mostly preserved for the largest K eigenvectors if the clusters of data are well separated. The idea is to select an initial solution in the form of K centroids, where each centroid is a row of $\boldsymbol{Y}$. Starting from any row of $\boldsymbol{Y}$, we repeatedly choose the next row that is closer to being perpendicular to all the centroids chosen so far. The angle between different rows can be trivially measured by means of the inner product; obviously we need to select such a row that the inner product with the rest of the centroids is as close to zero as possible.

6. Approximate Aggregation of Rate Matrices

In the general case, the partitioning algorithm will return a partition Δ over the state-space of C that only induces approximate equivalence of states (either in a QL or in an ASE context). The transitions of any component C have been described by a set of rate matrices $\{\boldsymbol{R}_a : a \in \vec{\mathcal{A}}(C)\}$, which correspond to the action types that C is capable of. Given a partition Δ, the aim is to produce a new set of rate matrices $\{\tilde{\boldsymbol{R}}_a : a \in \vec{\mathcal{A}}(\tilde{C})\}$ that characterises the activities of the aggregated $\tilde{C}$ component, whose states are labelled by the instances of Δ.

In this section, we investigate the effect of approximate aggregation on rate matrices. We distinguish between three different types of rate matrices, depending whether they have active or passive activities exclusively, or a mixture of the two. We clarify the impact of approximate aggregation in each case.

6.1. Real Rate Matrices

Let us first consider the case where no passive activities are involved, so for any action type a we have $\boldsymbol{R}_a \in \mathbb{R}^{N \times N}$, where N is the number of states in $ds(C)$. In Definition 5, approximate strong equivalence imposes that the class-to-class rates will be approximately the same for all of the states in the same class. In order to summarise these class-to-class transition rates with a single value, we simply use the average, which is a reasonable representative for populations that are nearly the same. More formally, given a component C and a partition $\Delta = \{A_1, \ldots, A_K\}$ on $ds(C)$ with K classes, then each instance of $\{\tilde{\boldsymbol{R}}_a : a \in \vec{\mathcal{A}}(C)\}$ will be a $K \times K$ rate matrix with entries:

$$\tilde{R}_{a,ij} = \frac{\sum_{k \in A_i} \sum_{l \in A_j} R_{a,kl}}{|A_i|} \tag{20}$$

where $|A_i|$ denotes the number of states included in class A_i.

6.2. Rate Matrices with Unspecified Rates

We now consider the case where a rate matrix describes passive activities; the entries of a rate matrix are exclusively either unspecified or zeros, that is $\boldsymbol{R}_a \in \{0, \top\}^{N \times N}$. It is possible to apply (20) in order to get an approximately lumped version of it.

Recall that an activity with rate $\top$ may be assigned a weight $w \in \mathbb{N}$, which determines the relative probability of that particular activity, while the absence of a weight simply implies that $w = 1$. An unspecified rate $\top$ should not be interpreted as "infinity", as it is important to keep track of its probability. Therefore, the process of aggregating a rate matrix $\boldsymbol{R}_a \in \{0, \top\}^{N \times N}$ is similar to the process of aggregating a real-valued rate matrix, as in the previous section. The effect of approximate aggregation will be on the weights that are associated with any passive rate $\top$, rather than on $\top$ itself.

Approximate aggregation as formulated in (20) involves two summations: given a state indexed by k we calculate the total rate of transitioning to a class A_j, and we average these total rates for every $k \in A_i$. The addition of two unspecified rates $w_1 \times \top$ and $w_2 \times \top$ is defined in [1] as follows:

$$w_1 \times \top + w_2 \times \top = (w_1 + w_2) \times \top \qquad \forall w_1, w_2 \in \mathbb{N} \tag{21}$$

However, in (20) we also have to divide the sum of the rates by the size of the outgoing class. Although division of an unspecified rate with a real number is not originally defined in [1], there is no reason why we cannot have such an operation. In fact, Smith [18] has defined multiplication of $\top$ by a real number c. Following Smith, we can more conveniently write:

$$\frac{w \times \top}{c} = \frac{w}{c} \times \top \tag{22}$$

If weights can be written as fractions as in (22), then the weight of an aggregated component could be a real rather than a natural number as required in [1]. Nevertheless, there should be no technical problem with considering weights that are positive real numbers. Regarding the behaviour of $\top$ in the context of the minimum operator, that will remain unchanged:

$$\min(w_1 \times \top, r) = \begin{cases} r, & r \in \mathbb{R}^+ \\ \min(w_1, w_2) \times \top, & r = w_2 \times \top \end{cases} \tag{23}$$

where $w_1, w_2 \in \mathbb{R}^+$.

In an approximate aggregation context, we allow w to be any positive real number. It can be easily seen that w will never be negative or zero; Equation (20) dictates that a weight w for an aggregated component will be the average of a set of values which are assumed to be positive.

6.3. Rate Matrices with both Active and Passive Activities

We demonstrate that rate matrices that contain a mixture of real-valued and unspecified rates are problematic. However, it is possible to avoid such an occurrence, if we impose certain restrictions on the model definition.

According to the original work on PEPA [1], a component C is not allowed to have an action type a that involves both active and passive activities. This requirement follows directly from the fact that the apparent rate $r_a(C)$ cannot be defined, since the addition of an active rate $r \in \mathbb{R}^+$ with an unspecified rate $\top$ has no clear semantic interpretation. Recall that the apparent rate $r_a(C)$ is the sum of all of the rates of the activities of type a in $\mathcal{A}ct(C)$. Therefore, it is important to impose that all of the activities of type a are consistently active or passive for the same C component.

The following is an example of a valid PEPA component which can be problematic:

$$\begin{aligned} C &\stackrel{def}{=} (a, r_1).C_1 + (a, r_2).C_2 \\ C_1 &\stackrel{def}{=} (b, r_3).C \\ C_2 &\stackrel{def}{=} (b, \top).C \end{aligned}$$

The C component is able to perform two actions a and b, each of which corresponds to a rate matrix:

$$\boldsymbol{R}_a = \begin{bmatrix} 0 & r_1 & r_2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \boldsymbol{R}_b = \begin{bmatrix} 0 & 0 & 0 \\ r_3 & 0 & 0 \\ \top & 0 & 0 \end{bmatrix}$$

Assuming a partition of the state-space $\Delta = \{\{C\}, \{C_1, C_2\}\}$, we can aggregate both matrices using (20) to obtain:

$$\tilde{\boldsymbol{R}}_a = \begin{bmatrix} 0 & r_1 + r_2 \\ 0 & 0 \end{bmatrix}, \qquad \tilde{\boldsymbol{R}}_b = \begin{bmatrix} 0 & 0 \\ 0.5 \times r_3 + 0.5 \times \top & 0 \end{bmatrix}$$

The aggregated rate matrix $\tilde{\boldsymbol{R}}_b$ is not valid, as there is no way to calculate the sum of the real rate $0.5 \times r_3$ with the unspecified rate $0.5 \times \top$. For the same reason, the apparent rate for action b cannot be defined. Therefore, the aggregated component is not a valid PEPA component, according to the original work on PEPA [1].

In order to overcome this problem, we have to impose an additional restriction. That is that for any PEPA component C, it should not allowed to have action types that involve both active and passive activities for the derivative set $ds(C)$. The C component of the previous example is not valid according to this specification. We can however rename the different occurrences of the b action type in such a way that we obtain the following valid component definitions:

$$\begin{aligned} C &\stackrel{def}{=} (a, r_1).C_1 + (a, r_2).C_2 \\ C_1 &\stackrel{def}{=} (b_1, r_3).C \\ C_2 &\stackrel{def}{=} (b_2, \top).C \end{aligned}$$

In the new version of C, there is consistency of active and passive actions over all of the states of the transition system. This is reflected in the following rate matrices that correspond to the actions that C, C_1 and C_2 are able to perform:

$$\boldsymbol{R}_a = \begin{bmatrix} 0 & r_1 & r_2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \boldsymbol{R}_{b_1} = \begin{bmatrix} 0 & 0 & 0 \\ r_3 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$\boldsymbol{R}_{b_2} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \top & 0 & 0 \end{bmatrix}$$

Given a partition $\Delta = \{\{C\}, \{C_1, C_2\}\}$, we can aggregate all three matrices using (20) to obtain:

$$\tilde{\boldsymbol{R}}_a = \begin{bmatrix} 0 & r_1 + r_2 \\ 0 & 0 \end{bmatrix}, \qquad \tilde{\boldsymbol{R}}_{b_1} = \begin{bmatrix} 0 & 0 \\ 0.5 \times r_3 & 0 \end{bmatrix},$$

$$\tilde{\boldsymbol{R}}_{b_2} = \begin{bmatrix} 0 & 0 \\ 0.5 \times \top & 0 \end{bmatrix}$$

To conclude, the rate matrix of a valid component C for some action a can be either $\boldsymbol{R}_a \in \mathbb{R}^{N \times N}$ or $\boldsymbol{R}_a \in \{0, \top\}^{N \times N}$. In both cases, Equation (20) will be used to produce the aggregated form for the transitions of type a.

7. Semantics for Aggregated Models

Our objective is to aggregate PEPA components individually and reflect this reduction on the CTMC that corresponds to a PEPA model containing these components. In this section we discuss an approach to produce compositions of aggregated components. Following the operational semantics of the language in Fig. 1, we define component aggregation and cooperation of aggregated components in terms of structured operational semantics.

7.1. Activity Aggregation

We shall formally describe the aggregated component in terms of the initial one; this will allow us to manipulate the operational semantics of PEPA in order to produce compositions of such components. We know that the labelled transition system of some component P corresponds to a collection of rate matrices for the different actions of P. In Section 6, we have discussed how we can aggregate these matrices given some partition of the state-space. Next we shall look into the interpretation of these new states and new transitions that have been produced.

States that belong to the same class will collapse into a single state. If Δ_P is a partition on P, then state aggregation can be formally described by the following rule:

$$\frac{A \in \Delta_P \quad P \in A}{P_A \equiv P} \tag{24}$$

Informally, the rule says that if A is a class in Δ_P, then there is a state P_A in the aggregated model which is interpreted as being in any of the states in the class. The aggregate state has been conveniently labelled with the class, however the interpretation would be no different with a different label. The important thing is that we miss the initial fine-grained information about the individual states in A. It is therefore inevitable to assume that all of the included states are equally probable.

The next step is to investigate how the activities are affected in the aggregated state-space. First, we consider the case of transitioning from an aggregated to a single state:

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P_A \xrightarrow{(\alpha,r/|A|)} P'} \ (P \in A)$$

Intuitively, the rule states that P_A is able to perform any of the activities of the components included in A. The rate however of each one of those activities is a fraction of their original rate. In fact, since we cannot discriminate between the states in A, all we know about P_A is that it is interpreted as P with probability $1/|A|$. Thus, whenever there is a rate r from $P \in A$ to P', the corresponding rate for P_A has to be adjusted accordingly.

Another rather simple case is when transitioning from a single state to an aggregated state. The rule that follows describes how an activity of a component P is distributed to a class $A' \in \Delta_P$.

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P \xrightarrow{(\alpha,r)} P_{A'}} \ (P' \in A')$$

According to the rule above, P is still able to perform the same activity at the same rate. The information about the exact target state may have been lost as we only know the target class, but this does not affect the rate of the activity.

Finally, the two cases above considered individually can be summarised in the following rule:

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P_A \xrightarrow{(\alpha,r/|A|)} P_{A'}} \ (P \in A, P' \in A') \tag{25}$$

If we apply the rule in (25) for every $P \in A$ and every $P' \in A'$, we essentially have multiple occurrences of the same activity, probably with different rates. This is totally acceptable as on the basis of the operational semantics PEPA models can be defined as multi-transition systems. It is reasonable however to collapse all of these activity instances into a single activity. Then the total rate from P_A to $P_{A'}$ for action type α will be:

$$q(P_A, P_{A'}, \alpha) = \frac{\sum_{P \in A} \sum_{P' \in A'} q(P, P', \alpha)}{|A|} \tag{26}$$

where $q(P, P', \alpha)$ is the rate of the activity of type α from P to P'. The rate expression above is equivalent to (20) used in the matrix-based aggregation of PEPA components in Section 6.

Regarding passive activities, these will be handled as discussed in Section 6. More specifically, we assume that any unspecified rate is associated with a weight $w > 0$. Moreover, approximate aggregation will have an effect on the weights, rather than on the corresponding unspecified rates. Finally, it is required that every action type is either globally active or globally passive for the derivative set of a given component.

7.2. The Aggregated Derivation Graph

So far, we have characterised component aggregation by a structured operational semantics approach. In this section, we investigate the effects that aggregation has on the PEPA semantics. We derive the aggregated semantics for each of the PEPA operators introduced in [1]. More specifically, we rewrite the operational semantics rules of Fig. 1 in the style that we have defined activities for aggregated components in (25). Eventually, we shall see that the aggregated semantics proposed are equivalent to the combined effect of the rule in (25) with the original PEPA semantics.

Prefix. According to the original specification of the prefix operator, whenever a component $(\alpha, r).P$ carries out an activity (α, r), it subsequently behaves as P. If now the target component is a class $A \in \Delta_P$, then the rate r should be distributed to the class:

$$\frac{(\alpha, r).P \xrightarrow{(\alpha,r)} P}{(\alpha, r).P_A \xrightarrow{(\alpha,r)} P_A} \; (P \in A) \tag{27}$$

The rule above describes the combined effect of (25) and the prefix rule in the original PEPA semantics.

Choice. The choice operator implies that a component $P + Q$ will behave either as P or Q. Let us first consider the case where there is an activity from P to P', assuming a partition Δ_P such that $A, A' \in \Delta_P$. Since the Q component is not selected, it does not matter whether it is aggregated or not. Regarding the P component, the activity in the aggregated state-space will be formed in a way similar to the rule in (25):

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P_A + Q \xrightarrow{(\alpha, r/|A|)} P_{A'}} \; (P \in A, P' \in A') \tag{28}$$

Similarly, whenever there is an activity from Q to Q', assuming a partition Δ_Q such that $B, B' \in \Delta_Q$, we have the following rule:

$$\frac{Q \xrightarrow{(\alpha,r)} Q'}{P + Q_B \xrightarrow{(\alpha, r/|B|)} Q_{B'}} \; (Q \in B, Q' \in B') \tag{29}$$

Unsynchronised Cooperation. An unsynchronised cooperation is defined over an action type a which does not belong in the cooperation set $\mathcal{L}$. Any component, either aggregated or not, may simply carry out any activity without affecting the state of other components. Therefore, starting from the rule in (25), we can consider the following rules for unsynchronised parallel composition:

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P_A \underset{L}{\bowtie} Q \xrightarrow{(\alpha,r)} P_{A'} \underset{L}{\bowtie} Q} \; (\alpha \notin L, P \in A, P' \in A') \tag{30}$$

$$\frac{Q \xrightarrow{(\alpha,r)} Q'}{P \underset{L}{\bowtie} Q_B \xrightarrow{(\alpha,r)} P \underset{L}{\bowtie} Q_{B'}} \; (\alpha \notin L, Q \in B, Q' \in B') \tag{31}$$

Synchronised Cooperation. Starting from the synchronised cooperation rule of Fig. 1, we shall modify it so as to admit aggregated components. Let Δ_P be a partition on P, where $A, A' \in \Delta_P$, and similarly Δ_Q be a partition on the state-space of Q, where $B, B' \in \Delta_Q$. We also assume that $P \in A$ and $P' \in A'$, while in the same way $Q \in B$ and $Q' \in B'$. Thus whenever there is a synchronised activity from P to P' and from Q to Q', for the corresponding aggregated states the activity will be defined as follows:

$$\frac{P \xrightarrow{(\alpha,r_1)} P' \quad Q \xrightarrow{(\alpha,r_2)} Q'}{P_A \underset{L}{\bowtie} Q_B \xrightarrow{(\alpha,R)} P_{A'} \underset{L}{\bowtie} Q_{B'}} \; (\alpha \in L, P \in A, P' \in A', Q \in B, Q' \in B') \tag{32}$$

where

$$R = \frac{r_1}{|A| \times r_\alpha(P_A)} \frac{r_2}{|B| \times r_\alpha(Q_B)} \min(r_\alpha(P_A), r_\alpha(Q_B)) \tag{33}$$

According to (25), we know that the rate of this particular activity for P_A will be $r_1/|A|$, while for Q_B it will be $r_2/|B|$. The rate expression above is identical to the original in Fig. 1, aside from the fact that we have modified the rates according to the aggregation rule in (25). The apparent rates can be easily derived; given an aggregated component P_A for some action type α, the apparent rate will be:

$$r_\alpha(P_A) = \sum_{A' \in \Delta_P} q(P_A, P_{A'}, \alpha) \tag{34}$$

We have already stated that an action type will be consistently active or passive for the entire derivative set of any component. The resulting apparent rate will be either a positive real number or unspecified, just as happens for non-aggregated components.

One last thing to consider is that the rule in (32) will be applied for every possible combination of states P, P', Q and Q' that belong to the classes A, A', B and B' correspondingly. The total rate of transitioning from $P_A \underset{L}{\bowtie} Q_B$ to $P_{A'} \underset{L}{\bowtie} Q_{B'}$ will be:

$$q(P_A \underset{L}{\bowtie} Q_B, P_{A'} \underset{L}{\bowtie} Q_{B'}, \alpha) = \\ \frac{\sum_{P \in A} \sum_{P' \in A'} q(P, P', \alpha)}{|A| \times r_\alpha(P_A)} \times \frac{\sum_{Q \in B} \sum_{Q' \in B'} q(Q, Q', \alpha)}{|B| \times r_\alpha(Q_B)} \times \min(r_\alpha(P_A), r_\alpha(Q_B))$$

Regarding the equation above, we observe that the first two terms aside from the apparent rates are equal to the collapsed activity rates for the aggregated components P_A and Q_A, as given by (26). So the total rate of a synchronised transition can be further simplified as follows:

$$q(P_A \underset{L}{\bowtie} Q_B, P_{A'} \underset{L}{\bowtie} Q_{B'}, \alpha) = \\ \frac{q(P_A, P_{A'}, \alpha)}{r_\alpha(P_A)} \times \frac{q(Q_B, Q_{B'}, \alpha)}{r_\alpha(Q_B)} \times \min(r_\alpha(P_A), r_\alpha(Q_B)) \tag{35}$$

The rate expression for aggregated components in (35) is essentially identical to the expression used for non-aggregated synchronised activities. Therefore, activity aggregation as defined in Section 7.1 allows us to employ the operational semantics in Fig. 1 to construct the derivation graph for any composition of aggregated components.

Hiding. Recall that a component P/L behaves as P, except that any activity of any action type in L will be hidden, meaning that it will be considered as a local activity. Combining the semantics of rules for the hiding operator in Fig. 1 with the rule in (25), we obtain the following operators that describe the combined effect:

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P_A/L \xrightarrow{(\alpha,r/|A|)} P_{A'}/L} \; (\alpha \notin L, P \in A, P' \in A') \tag{36}$$

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P_A/L \xrightarrow{(\tau,r/|A|)} P_{A'}/L} \; (\alpha \in L, P \in A, P' \in A') \tag{37}$$

7.3. A Worked Example

We shall now illustrate the concepts discussed in this section with an example. Let us consider the composition $P_1 \underset{\{a,b\}}{\bowtie} Q_1$ of the following components:

$$\begin{array}{lcl} P_1 & \stackrel{def}{=} & (c,r).P_2 + (a,\top).P_3 \\ P_2 & \stackrel{def}{=} & (c,r).P_1 \\ P_3 & \stackrel{def}{=} & (c,r).P_4 \\ P_4 & \stackrel{def}{=} & (c,r).P_3 + (b,\top).P_2; \end{array} \qquad \begin{array}{lcl} Q_1 & \stackrel{def}{=} & (a,r_a).Q_2 \\ Q_2 & \stackrel{def}{=} & (b,r_b).Q_1 \end{array}$$

We assume a partition $\Delta = \{\{P_1, P_2\}, \{P_3, P_4\}\}$ on the state-space of P_1. Applying the standard PEPA semantics, for the P_1 component only, will result in the graph of Fig. 3.

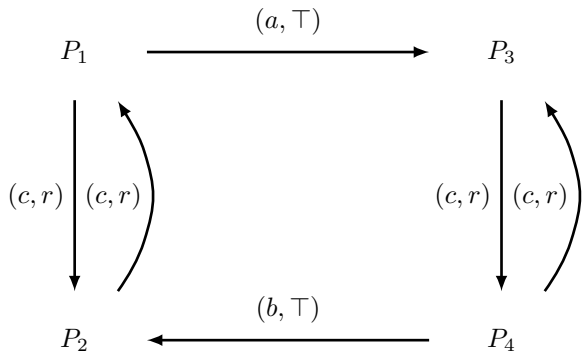


Figure 3: Derivation graph for P_1

It is preferable to aggregate the state-space of P_1 first, and then apply the rules for the composition with the Q_1 component. In this way, we avoid unnecessarily large intermediate components. According to Δ, the states P_1 and P_2 are collapsed into a single state; let that be P_{12}. Similarly, P_3 and P_4 are collapsed into P_{34}.

The next step will be to apply the rule in (25) for states P_1 and P_2, and their amalgamation P_{12}. This will add three activities in the aggregated derivation graph: $(c, r/2).P_{12}$, $(c, r/2).P_{12}$ and $(a, \top/2).P_{34}$. Similarly, applying rule (25) on P_3 and P_4 will add the activities: $(c, r/2).P_{34}$, $(c, r/2).P_{34}$ and $(b, \top/2).P_{12}$. The aggregated derivation graph for the component is pictured in Fig. 4. Activities of the same type having the same source and target states can be collapsed into a single activity whose rate is the sum of the corresponding activity rates.

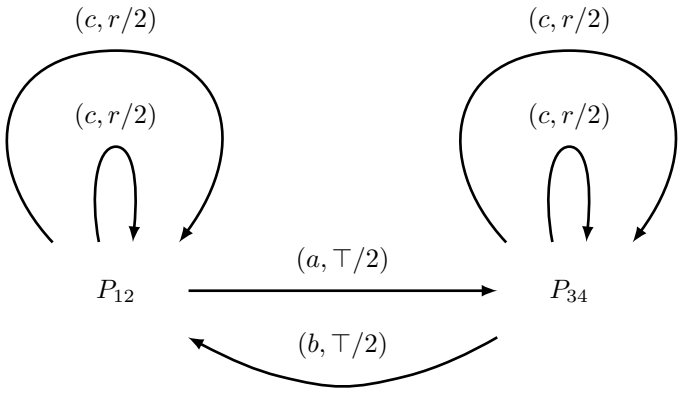


Figure 4: Derivation graph for P_{12}

Finally, using the standard operational semantics of PEPA, we can create the state-space for the composition $P_1 \underset{\{a,b\}}{\bowtie} Q_1$ as depicted in Fig. 5. It is easy to see that the same graph would have been produced if we first derive the full graph for $P_1 \underset{\{a,b\}}{\bowtie} Q_1$, and subsequently apply the aggregated operational semantics of Section 7.2.

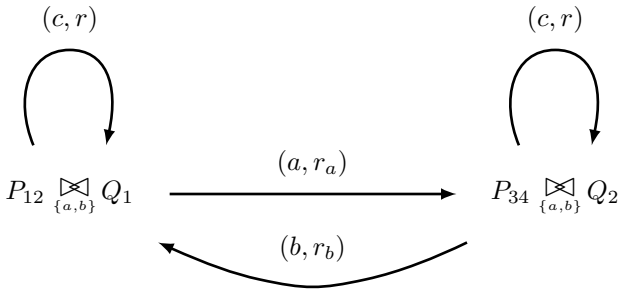


Figure 5: Derivation graph for $P_{12} \underset{\{a,b\}}{\bowtie} Q_1$

7.4. Side-Effects of Approximate Aggregation

By abstracting every state P that belongs to some class A with a single aggregated state P_A, it has been possible to construct components of smaller size. This aggregation has been characterised as approximate, since it essentially relies on a partition Δ that only induces an approximately strongly equivalent aggregated component. In this section, we examine some implications of this approximate aggregation on the state-space and the activities of the original model.

The first thing to consider is what information the structure of an aggregated component P' conveys with respect to the original component P. Regarding interpretation of aggregated states, we already know from the rule in (24) that an aggregated state P_A is interpreted as being in any of the states in the set A. The next thing to examine is whether the set of reachable states is affected by aggregation. The aggregation rule in (25) guarantees that all the activities in the original component will be represented by aggregated activities in the aggregated component. That means that the reachable set for an aggregated component will be no smaller than the original reachable set. The converse is not true, as we have the following proposition:

Proposition 1. *Let P_A and P'_A be aggregated components that represent the classes $A, A' \in \Delta$ correspondingly. If there is an activity of the form:*

$$P_A \xrightarrow{(\alpha,r)} P_{A'}$$

then in terms of the original model this is interpreted as:

$$P \xrightarrow{(\alpha,r|A|)} P', \quad \forall P \in A,\ \forall P' \in A'$$

The proof of Proposition 1 follows easily if we invert the effect of the rule in (25), which defines activity aggregation. Intuitively, an aggregated activity of the form $(\alpha, r|A|)$ is introduced for any pair P, P', even if that was not existent in the first place. In other words, approximate aggregation may introduce forged activities in terms of the original component.

The existence of forged activities has also an effect in the case of synchronised parallel composition. More specifically, there is no guarantee that a reachable state in the original model will also be reachable in the aggregated model as well. We shall demonstrate through an example that approximate aggregation may introduce deadlocks in the case of synchronised cooperation of aggregated components. Let $P_1 \underset{\{a,b\}}{\bowtie} Q_1$ be the composition of the following components:

$$
\begin{array}{rcl}
P_1 & \stackrel{def}{=} & (a, r).P_2 \\
P_2 & \stackrel{def}{=} & (a, r).P_3 \\
P_3 & \stackrel{def}{=} & (b, r).P_1
\end{array}
\qquad\qquad
\begin{array}{rcl}
Q_1 & \stackrel{def}{=} & (a, r).Q_2 \\
Q_2 & \stackrel{def}{=} & (a, r).Q_3 \\
Q_3 & \stackrel{def}{=} & (b, r).Q_1
\end{array}
$$

In short, both of the components above perform two synchronised activities of type a in succession, and then they perform a synchronised activity of type b, returning to their initial states. Therefore, the reachable set will consist of the states: $P_1 \underset{\{a,b\}}{\bowtie} Q_1$, $P_2 \underset{\{a,b\}}{\bowtie} Q_2$ and $P_3 \underset{\{a,b\}}{\bowtie} Q_3$. Assuming a partition $\Delta = \{\{P_1, P_2\}, \{P_3\}\}$ on the state-space of P_1, then the aggregated version P_{12} will be as follows:

$$
\begin{array}{rcl}
P_{12} & \stackrel{def}{=} & (a, r).P_{12} + (a, r).P_3 \\
P_3 & \stackrel{def}{=} & (b, r).P_{12}
\end{array}
$$

The self-loop introduced in P_{12} will effectively change the dynamics of the system. Following Proposition 1, we can produce a reconstruction of P_{12} in the original state-space:

$$
\begin{array}{rcl}
P_1' & \stackrel{def}{=} & (a, 2r).P_2' + (a, 2r).P_3 \\
P_2' & \stackrel{def}{=} & (a, 2r).P_1' + (a, 2r).P_3 \\
P_3 & \stackrel{def}{=} & (b, r).P_1'
\end{array}
$$

Regarding P_1', there is nothing that enforces the succession of two activities of type a before entering the P_3 state. In fact, the P_1' component, and as a consequence P_{12} as well, may jump ahead to P_3 with a single activity of type a, and then wait for a b activity. The Q component however still requires two type a activities before it is able to offer a type b. We see that in this example approximate aggregation has effectively introduced a deadlock.

In the general case, we can provide no guarantees regarding the reachable states of an approximately aggregated model. We have seen that for individual components, if a state has been reachable in the original component state-space, it will also be reachable in the aggregated state-space, however the converse is not true, since forged activities are introduced. In the case of synchronised cooperation, forged activities may also result in deadlocks, meaning that part of the original state-space may no longer be reachable.

8. A Case Study on Cloud Computing

Cloud systems typically consist of many inhomogeneous components, whose interactions define a complex behaviour. The PEPA language offers a framework to represent parts of a system in isolation, while larger components are formed as compositions of simpler building blocks. However, increased complexity often renders analysis difficult, which makes it an ideal example to demonstrate the potential of approximation techniques. The problem under investigation is the scalability of different routing policies in a *Platform as a Service* (PaaS) system. In short, a routing algorithm is responsible for directing the workload to a number of leased servers. This case study has been inspired by an issue that has been raised for the Heroku PaaS provider.

According to the on-line Heroku specification documents,[1] a dyno is a lightweight environment running a single command at a time. This functionality is implemented by an isolated virtualised server. There are two kinds of dynos available: *web dynos* which respond to HTTP requests, and *worker dynos* which execute background jobs. Commands are not supposed to be interrupted, thus concurrency is achieved by employing more than one dyno. Increasing the number of web dynos will increase the concurrency of HTTP requests, while more worker dynos provide more capacity for processes running in the background. Therefore, all the

[1] https://devcenter.heroku.com/

Table 1: The rate values used in the examples

Variable Name	Value (sec^{-1})
$r_{request}$	[40, 60]
r_{web}	8
$r_{migrate}$	1
r_{worker}	4
$r_{response}$	20
r_{assign}	500

client has to do is to upload the source code of the application and scale it to a number of dynos. The idea is that once a service request appears, Heroku will be responsible for assigning that request to one of the dynos that have been leased by the client, by following a routing policy. The routing policy is the key component that we shall investigate in this case study. We list below two routing policies that have historically been used by Heroku:

Random Routing implies that a new request is directed to a randomly-selected web dyno. The premise of random routing is that the load is balanced across the dynos in the long term.

Smart Routing involves tracking the availability of each dyno and the load is directed accordingly, thus minimising the number of dynos that remain idle.

Although explicit information on the implementation of these policies is not available, it is straightforward to model the desired behaviour for each policy at a high-level.

Rap Genius[2] is a website that aims to provide a critical and artistic insight into the lyrics of rap songs. The website users have access to content via HTTP requests, and they are able to add annotations to content. Rap Genius makes this service available via the cloud, and Heroku in particular. In the beginning of 2013, Rap Genius reported unusually long average response times, despite the large number of dynos leased by the website[3]. This performance has been attributed to a change in the Heroku routing policy from smart to random.

Our objective is not to assess the quality of service provided by Heroku, or recreate absolutely realistic expectations of the response time for Rap Genius. Instead, we want to demonstrate how modelling with Markov chains may capture the effect of different routing policies, and most importantly, whether compositional aggregation produces results accurate enough to reach the same conclusions in a more efficient way.

8.1. Modelling Routing Policies with PEPA

The activity types and the rates in our model are going to be the same, regardless of the routing policy. Table 1 summarises the rates of the events considered. The number of requests will be governed by a Poison process with rate $r_{request}$. We experiment with two different workloads, 40 to 60 sec^{-1}, which correspond to 2400 and 3600 requests per minute correspondingly.

8.1.1. Random Routing Policy

A dyno can be anything between idle, occupied or having one or more requests in its local queue. According to the random routing policy, a router is supposed to randomly assign jobs to dynos, regardless of their state. In terms of PEPA models, and subsequently CTMCs, it is very simple to represent such a probabilistic behaviour.

Web dynos are represented by components $WebDyno_i$, where the subscript i denotes the number of requests in the local dyno queue. For $WebDyno_i$, three activities are possible; *service* realises the main web

[2] http://rapgenius.com/

[3] http://rapgenius.com/James-somers-herokus-ugly-secret-lyrics

service part, whose completion proceeds to the response stage, carried out by $WebDyno_{ia}$. It is assumed that a response cannot be interrupted, thus no new job can be assigned or enqueued at this point. Given that the response rate is significantly higher than the web service rate (see Table 1), this should not affect the availability of the dyno. The *migrate* activity generates a migration request and decreases the queue length. Finally, the $assign_{web}$ activity adds a request to the queue.

$$\begin{array}{rcl} WebDyno & \stackrel{def}{=} & (assign_{web}, \top).WebDyno_0 \\ WebDyno_i & \stackrel{def}{=} & (service, r_{web}).WebDyno_{ia} \\ & + & (migrate, r_{migrate}).WebDyno_{i-1} \\ & + & (assign_{web}, \top).WebDyno_{i+1} \\ WebDyno_{ia} & \stackrel{def}{=} & (response, r_{response}).WebDyno_{i-1} \end{array}$$

The components $WebDyno_i$ and $WebDyno_{ia}$ represent the two stages of a web service. In both cases, the web dyno is considered to be occupied. The idle state is denoted by $WebDyno$, which only performs an $assign_{web}$ activity.

Regarding the worker dynos, they have a similar but simpler structure, as there is no job migration option in this case. Other than that, assignment has been labelled by a distinct activity $assign_{worker}$, in order to distinguish between the two possible types of assignments.

$$\begin{array}{rcl} WorkerDyno & \stackrel{def}{=} & (assign_{worker}, \top).WorkerDyno_0 \\ WorkerDyno_i & \stackrel{def}{=} & (service, r_{worker}).WorkerDyno_{ia} \\ & + & (assign_{worker}, \top).WorkerDyno_{i+1} \\ WorkerDyno_{ia} & \stackrel{def}{=} & (response, r_{response}).WorkerDyno_{i-1} \end{array}$$

The routing component is characterised by a set of states that denote the number of requests in the router queue. At any state, the router should be able to accept a web request or a migration request, either of which will be added in the router queue. Remember that web requests are modelled by a Poisson process, thus the *request* activity has a constant rate for any state of the router component. When a new job arrives, the router will attempt to direct it to any of the web or worker dynos, depending on the type of the request. It is more convenient to model the router as two queues, one for each type of dyno. So for the web requests we have:

$$\begin{array}{rcl} WebRouter_0 & \stackrel{def}{=} & (request, r_{request}).WebRouter_1 \\ WebRouter_i & \stackrel{def}{=} & (request, r_{request}).WebRouter_{i+1} \\ & + & (assign_{web}, r_{assign}).WebRouter_{i-1} \\ WebRouter_n & \stackrel{def}{=} & (request, r_{request}).WebRouter_n \\ & + & (assign_{web}, r_{assign}).WebRouter_{n-1} \end{array}$$

where n denotes the maximum size for the corresponding queue. If the maximum size is reached, it is assumed that any new request will be discarded until the queue is not full. In the same way, for migration requests we have:

$$\begin{array}{rcl} WorkerRouter_0 & \stackrel{def}{=} & (migrate, \top).WorkerRouter_1 \\ WorkerRouter_i & \stackrel{def}{=} & (migrate, \top).WorkerRouter_{i+1} \\ & + & (assign_{worker}, r_{assign}).WorkerRouter_{i-1} \\ WorkerRouter_n & \stackrel{def}{=} & (migrate, \top).WorkerRouter_n \\ & + & (assign_{worker}, r_{assign}).WorkerRouter_{n-1} \end{array}$$

Finally, the router will be the parallel composition of the components above.

Figure 6 outlines the complete model of the random routing policy. Note that the model imposes a maximum dyno queue length equal to 1. As we shall see later in Section 8.2, this model is adequate to observe the qualitative difference between a random and a smart routing policy.

8.1.2. Smart Routing Policy

The modelling of dynos does not substantially differ from that of the random routing case, as both web and worker dynos are characterised by the same states and the same rates. The only difference is that we now have two distinct action types for directing a job to a dyno. We want to capture the fact that a job may be either assigned to an idle dyno, or enqueued to an occupied dyno. Regarding the web dynos, only a

$$
\begin{array}{rcl}
WebDyno & \stackrel{def}{=} & (assign_{web}, \top).\, WebDyno_0 \\
WebDyno_0 & \stackrel{def}{=} & (service, r_{web}).\, WebDyno_{0a} + (migrate, r_{migrate}).\, WebDyno \\
 & + & (assign_{web}, \top).\, WebDyno_1 \\
WebDyno_{0a} & \stackrel{def}{=} & (response, r_{response}).\, WebDyno \\
WebDyno_1 & \stackrel{def}{=} & (service, r_{web}).\, WebDyno_{1a} + (migrate, r_{migrate}).\, WebDyno_0 \\
WebDyno_{1a} & \stackrel{def}{=} & (response, r_{response}).\, WebDyno_0
\end{array}
$$

$$
\begin{array}{rcl}
WorkerDyno & \stackrel{def}{=} & (assign_{worker}, \top).\, WorkerDyno_0 \\
WorkerDyno_0 & \stackrel{def}{=} & (service, r_{worker}).\, WorkerDyno_{0a} + (assign_{worker}, \top).\, WorkerDyno_1 \\
WorkerDyno_{0a} & \stackrel{def}{=} & (response, r_{response}).\, WorkerDyno \\
WorkerDyno_1 & \stackrel{def}{=} & (service, r_{worker}).\, WorkerDyno_{1a} \\
WorkerDyno_{1a} & \stackrel{def}{=} & (response, r_{response}).\, WorkerDyno_0
\end{array}
$$

$$
\begin{array}{rcl}
WebRouter_0 & \stackrel{def}{=} & (request, r_{request}).\, WebRouter_1 \\
WebRouter_1 & \stackrel{def}{=} & (request, r_{request}).\, WebRouter_2 + (assign_{web}, r_{assign}).\, WebRouter_0 \\
WebRouter_2 & \stackrel{def}{=} & (request, r_{request}).\, WebRouter_3 + (assign_{web}, r_{assign}).\, WebRouter_1 \\
WebRouter_3 & \stackrel{def}{=} & (request, r_{request}).\, WebRouter_3 + (assign_{web}, r_{assign}).\, WebRouter_2
\end{array}
$$

$$
\begin{array}{rcl}
WorkerRouter_0 & \stackrel{def}{=} & (migrate, \top).\, WorkerRouter_1 \\
WorkerRouter_1 & \stackrel{def}{=} & (migrate, \top).\, WorkerRouter_2 + (assign_{worker}, r_{assign}).\, WorkerRouter_0 \\
WorkerRouter_2 & \stackrel{def}{=} & (migrate, \top).\, WorkerRouter_3 + (assign_{worker}, r_{assign}).\, WorkerRouter_1 \\
WorkerRouter_3 & \stackrel{def}{=} & (migrate, \top).\, WorkerRouter_3 + (assign_{worker}, r_{assign}).\, WorkerRouter_2
\end{array}
$$

$$
Random_{N:M} \stackrel{def}{=} WebDyno[N] \parallel WorkerDyno[M] \underset{\mathcal{L}_{random}}{\bowtie} (WebRouter_0 || WorkerRouter_0)
$$

where $\mathcal{L}_{random} = \{assign_{web}, assign_{worker}, migrate\}$

Figure 6: PEPA model for random Heroku routing

WebDyno component will now be able to perform an $assign_{web}$ activity, as it denotes that the dyno is idle. For a $WebDyno_i$ component, which denotes an occupied web dyno with i requests in its local queue, jobs can only be enqueued as follows:

$$
\begin{array}{rcl}
WebDyno & \stackrel{def}{=} & (assign_{web}, \top).\, WebDyno_0 \\
WebDyno_i & \stackrel{def}{=} & (service, r_{web}).\, WebDyno_{ia} \\
 & + & (migrate, r_{migrate}).\, WebDyno_{i-1} \\
 & + & (enqueue_{web}, \top).\, WebDyno_{i+1}
\end{array}
$$

The choice between assignment or placement in the local queue is a responsibility of the routing policy. Similarly, an $assign_{worker}$ activity can only be performed by *WorkerDyno*, while for the $WorkerDyno_i$ component we have:

$$
\begin{array}{rcl}
WorkerDyno & \stackrel{def}{=} & (assign_{worker}, \top).\, WorkerDyno_0 \\
WorkerDyno_i & \stackrel{def}{=} & (service, r_{worker}).\, WorkerDyno_{ia} \\
 & + & (enqueue_{worker}, \top).\, WorkerDyno_{i+1}
\end{array}
$$

The smart routing policy consists of simply directing a request to a dyno that is available. If more than one dyno is available, then the router will randomly select a dyno. If there are no dynos available at a certain moment, then the request will be randomly enqueued to any dyno. The routing algorithm involves a deterministic step, which is the dyno availability check. Such a deterministic behaviour cannot be directly modelled according to the standard definition of PEPA. What we can do instead is to probabilistically favour assigning jobs to free dynos rather than placing then in queues. The idea is that the router will delay

directing a request until a dyno is available. This delay should not be infinite however; if too many requests arrive, then the router will decrease its queue length by directing the requests to random dynos.

Regarding the *WebRouter* component, let n be the maximum queue length. Then for any queue length $i < n$, the requests will be assigned to web dynos that can perform an $assign_{web}$ activity; that means that the dyno in question is idle.

$$\begin{array}{rcl} WebRouter_0 & \stackrel{def}{=} & (request, r_{request}).WebRouter_1 \\ WebRouter_i & \stackrel{def}{=} & (request, r_{request}).WebRouter_{i+1} \\ & + & (assign_{web}, r_{assign}).WebRouter_{i-1} \end{array}$$

If the queue length reaches its maximum size n, that probably means that no dyno has been available for a long time; it is then acceptable to send the request to the queue of any dyno. Let $WebRouter_n$ represent the state at which the corresponding router queue is full. Then $WebRouter_n$ will either assign or enqueue a request.

$$\begin{array}{rcl} WebRouter_n & \stackrel{def}{=} & (request, r_{request}).WebRouter_n \\ & + & (assign_{web}, r_{assign} \times 0.5).WebRouter_{n-1} \\ & + & (enqueue_{web}, r_{assign} \times 0.5).WebRouter_{n-1} \end{array}$$

By using similar reasoning, the migration queue on the router side will be modified as follows:

$$\begin{array}{rcl} WorkerRouter_0 & \stackrel{def}{=} & (migrate, \top).WorkerRouter_1 \\ WorkerRouter_i & \stackrel{def}{=} & (migrate, \top).WorkerRouter_{i+1} \\ & + & (assign_{worker}, r_{assign}).WorkerRouter_{i-1} \end{array}$$

$$\begin{array}{rcl} WorkerRouter_n & \stackrel{def}{=} & (migrate, \top).WorkerRouter_n \\ & + & (assign_{worker}, r_{assign} \times 0.5).WorkerRouter_{n-1} \\ & + & (enqueue_{worker}, r_{assign} \times 0.5).WorkerRouter_{n-1} \end{array}$$

where n denotes the maximum queue length, and $i < n$.

To summarise, when the queue on the router part is not full, then the router works according to its "smart" mode of operation; it directs any requests to idle dynos only. Any new requests will have to wait in the router queue before being assigned. However, if the router queue reaches maximum capacity, this is an indication that the system is congested, suggesting that there are no idle dynos available. The router will then enter its "random" mode of operation, as it will decrease its queue by randomly directing requests to any dyno. That is captured by the fact that $enqueue_{web}$ and $enqueue_{worker}$ can only be performed if the corresponding router queue is full. The complete model for the smart routing policy is shown in Fig. 7.

8.2. Evaluation of Routing Policies

In this section, we experimentally evaluate how the routing policies considered respond to different workloads. We consider a system whose components are appropriate for compositional aggregation, and we evaluate the effect of aggregation approaches on the system behaviour.

More specifically, we consider a system featuring 8 web dynos and 8 worker dynos. We have two models that implement the two routing policies; these are $Random_{8:8}$ and $Smart_{8:8}$. By approximately reducing the components *WebDyno*[8] and *WorkerDyno*[8] to 60% of their original size each, the global state-space is effectively reduced to 36%. The original and the aggregated versions of the models have been solved for their transient and steady-state behaviour via the sparse engine of the PRISM model checker [19]. The steady-state distribution has been calculated using the Gauss-Seidel method. The transient probabilities have been calculated via the uniformisation method. The experiments have been performed in an Intel® Xeon™ E5430 @ 2.66GHz PC running Ubuntu Linux.

The running times for $Random_{8:8}$ and $Smart_{8:8}$ are summarised in Tables 2 and 3 correspondingly. According to these tables, the state-space reduction resulted in the expected reduction in the analysis time, for both approximate aggregation methods, based on quasi-lumpability (QL) and approximate strong equivalence (ASE) correspondingly. The most interesting thing regarding the running times is that the time needed to approximate the state-space is trivial compared to the time saved.

$$
\begin{aligned}
WebDyno &\stackrel{def}{=} (assign_{web}, \top).WebDyno_0 \\
WebDyno_0 &\stackrel{def}{=} (service, r_{web}).WebDyno_{0a} + (migrate, r_{migrate}).WebDyno \\
&+ (enqueue_{web}, \top).WebDyno_1 \\
WebDyno_{0a} &\stackrel{def}{=} (response, r_{response}).WebDyno \\
WebDyno_1 &\stackrel{def}{=} (service, r_{web}).WebDyno_{1a} + (migrate, r_{migrate}).WebDyno_0 \\
WebDyno_{1a} &\stackrel{def}{=} (response, r_{response}).WebDyno_0
\end{aligned}
$$

$$
\begin{aligned}
WorkerDyno &\stackrel{def}{=} (assign_{worker}, \top).WorkerDyno_0 \\
WorkerDyno_0 &\stackrel{def}{=} (service, r_{worker}).WorkerDyno_{0a} + (enqueue_{worker}, \top).WorkerDyno_1 \\
WorkerDyno_{0a} &\stackrel{def}{=} (response, r_{response}).WorkerDyno \\
WorkerDyno_1 &\stackrel{def}{=} (service, r_{worker}).WorkerDyno_{1a} \\
WorkerDyno_{1a} &\stackrel{def}{=} (response, r_{response}).WorkerDyno_0
\end{aligned}
$$

$$
\begin{aligned}
WebRouter_0 &\stackrel{def}{=} (request, r_{request}).WebRouter_1 \\
WebRouter_1 &\stackrel{def}{=} (request, r_{request}).WebRouter_2 + (assign_{web}, r_{assign}).WebRouter_0 \\
WebRouter_2 &\stackrel{def}{=} (request, r_{request}).WebRouter_3 + (assign_{web}, r_{assign}).WebRouter_1 \\
WebRouter_3 &\stackrel{def}{=} (request, r_{request}).WebRouter_3 \\
&+ (assign_{web}, r_{assign} \times 0.5).WebRouter_2 \\
&+ (enqueue_{web}, r_{assign} \times 0.5).WebRouter_2
\end{aligned}
$$

$$
\begin{aligned}
WorkerRouter_0 &\stackrel{def}{=} (migrate, \top).WorkerRouter_1 \\
WorkerRouter_1 &\stackrel{def}{=} (migrate, \top).WorkerRouter_2 + (assign_{worker}, r_{assign}).WorkerRouter_0 \\
WorkerRouter_2 &\stackrel{def}{=} (migrate, \top).WorkerRouter_3 + (assign_{worker}, r_{assign}).WorkerRouter_1 \\
WorkerRouter_3 &\stackrel{def}{=} (migrate, \top).WorkerRouter_3 \\
&+ (assign_{worker}, r_{assign} \times 0.5).WorkerRouter_2 \\
&+ (enqueue_{worker}, r_{assign} \times 0.5).WorkerRouter_2
\end{aligned}
$$

$$
Smart_{N:M} \stackrel{def}{=} WebDyno[N] \parallel WorkerDyno[M] \underset{\mathcal{L}_{smart}}{\bowtie} (WebRouter_0 || WorkerRouter_0)
$$

$$
\mathcal{L}_{smart} = \{assign_{web}, enqueue_{web}, assign_{worker}, enqueue_{worker}, migrate\}
$$

Figure 7: PEPA model for smart Heroku routing

Table 2: Running Times for $Random_{8:8}$

	Original	QL-based Aggregation	ASE-based Aggregation
Approximation	-	3 sec	3 sec
PRISM Loading	10000 sec	4000 sec	3000 sec
Transient Solution[a]	29000 sec	13000 sec	9000 sec
Steady-State solution	250 sec	200 sec	100 sec
Total Time	39250 sec	17200 sec	12100 sec
Number of states	3920400	1411344	1411344

[a] 50 points: $0 \leq t \leq 4$

The next thing to see is whether the approximate results obtained provide us with reliable information regarding the properties of the routing policies. We have experimented with two different values for the

Table 3: Running Times for $Smart_{8:8}$

	Original	QL-based Aggregation	ASE-based Aggregation
Approximation	-	3 sec	3 sec
PRISM Loading	11000 sec	5000 sec	5000 sec
Transient Solution[b]	32000 sec	14000 sec	15000 sec
Steady-State solution	370 sec	230 sec	300 sec
Total Time	43370 sec	19230 sec	20300 sec
Number of states	3849444	1401852	1397124

[b] 50 points: $0 \leq t \leq 4$

request rate 40 and 60, in order to observe how the two routing policies respond to different workloads. The effects of each policy should be reflected in the average dyno queue length and in the number of dynos that remain idle. As a general remark on the results that follow, the ASE-based partitioning strategy provides much more accurate results than the QL-based method.

Figure 8 outlines the transient behaviour for request arrival rate equal to 40 sec^{-1}, or 2400 requests per minute. The data plotted depicts how the average population of idle dynos and average local queue lengths change during the first four seconds of the system being online. We can see that after these four seconds, the system appears to be in steady state. The left column of plots presents results for the random routing policy, while the plots on the right column correspond to smart routing.

Judging by the first two plots in Figs. 8(a) and 8(b), which show the results of the original model, part of the system is underused for both smart and random routing, as we have a significant number of idle dynos in both cases. However, the average dyno queue lengths are considerably higher for random routing. This means that some requests might have to wait in the queue, while there are dynos available. That is not the case for smart routing however, where the dyno queues are almost empty. In other words, the smart routing fully exploits the capacity of the Heroku configuration, in contrast with the random routing policy.

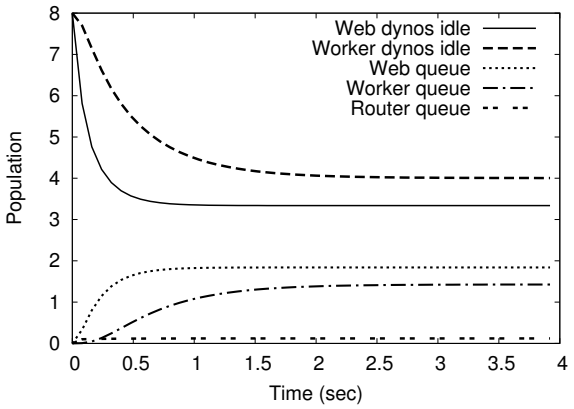
Regarding the results of the QL-based method in Figs. 8(c) and 8(d), they seem to qualitatively agree with the true results, although the numerical values for the average idle dynos do not exactly match. Figures 8(e) and 8(f) summarise the results for the ASE-based aggregation, where the errors appear to be significantly smaller. This can be also verified in Fig. 9, which depicts the relative errors in measuring the queue lengths for the partitioning strategies and the routing policies considered. The relative error at time t is calculated as follows:

$$\eta(t) = \frac{|f(t) - \tilde{f}(t)|}{\max\limits_{t} f(t)} \tag{38}$$

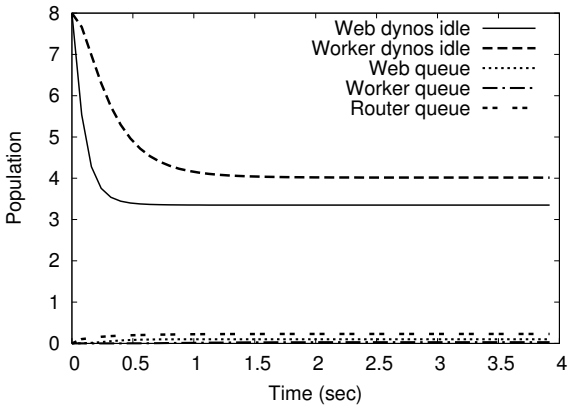
where $f(t)$ is the true value for the queue length at time t, and $\tilde{f}(t)$ is the corresponding approximate value. We think that a modest reduction in accuracy was a worthwhile price to pay, especially given the substantial reduction in analysis time (Tables 2 and 3). For ASE-based aggregation, the errors appear to be insignificant.

In the experiment summarised in Fig. 10, we investigate how the routing policies are affected by a higher workload, by increasing the request arrival rate to 60 sec^{-1}, or 3600 requests per minute. According to Figs. 10(a) and 10(b), which reflect the behaviour of the unreduced models, the system usage is similar for both random and smart routing, as we can see by the numbers of idle web and worker dynos. For the smart system, the dyno queues have significantly shorter length when compared to the random routing policy, implying that the requests wait less time until they are serviced. As a final comment, we can say that a request arrival rate to 40 is probably the most that the smart routing policy can effectively handle for the number of dynos considered.

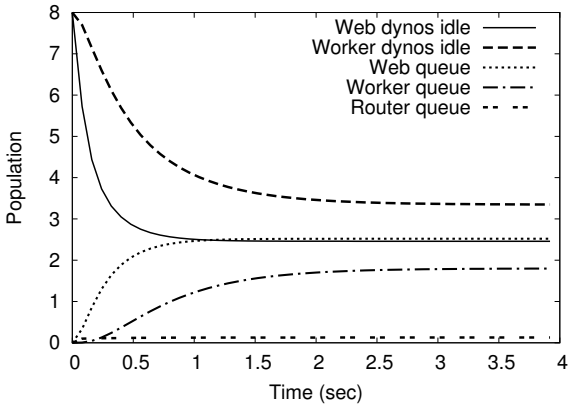
The results of the QL-based approach in Figs. 10(c) and 10(d) give a similar picture. The relation



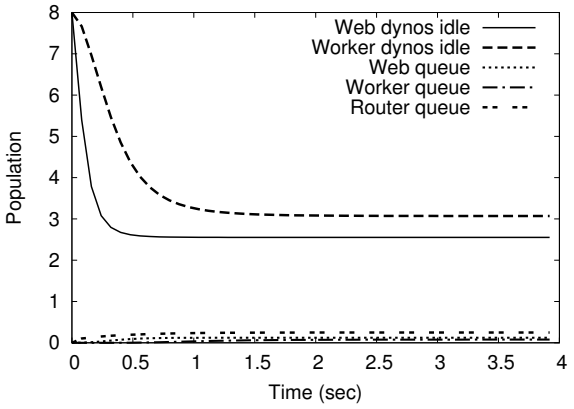
(a) Random routing (Original)



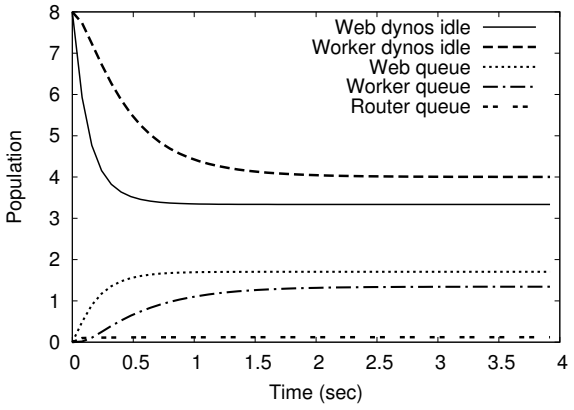
(b) Smart routing (Original)



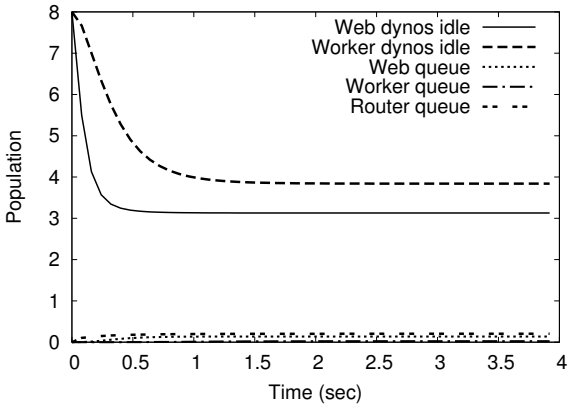
(c) Random routing (QL-based aggregation)



(d) Smart routing (QL-based aggregation)

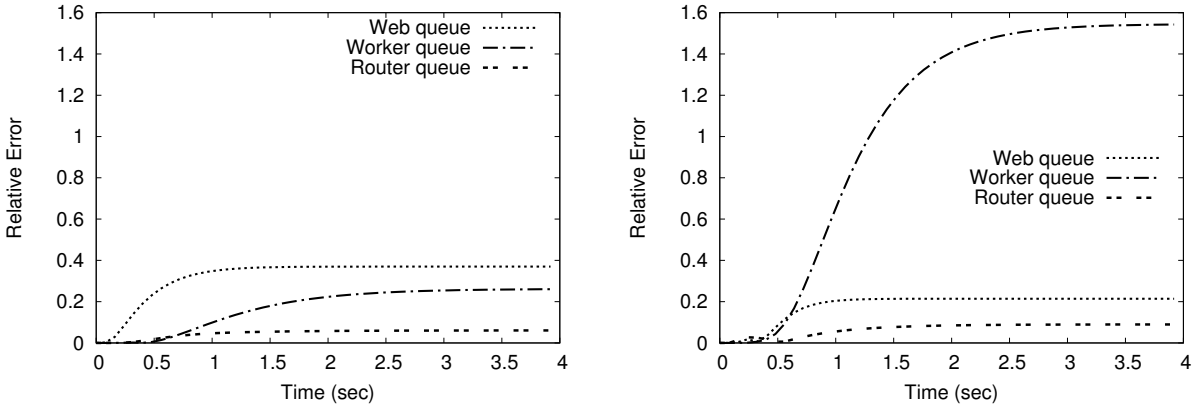


(e) Random routing (ASE-based aggregation)



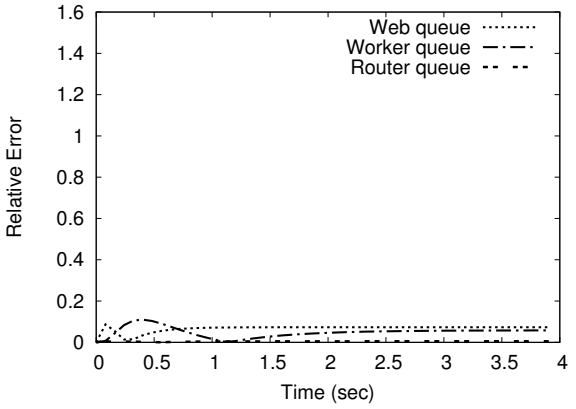
(f) Smart routing (ASE-based aggregation)

Figure 8: $Random_{8:8}$ and $Smart_{8:8}$ results for $r_{request} = 40$

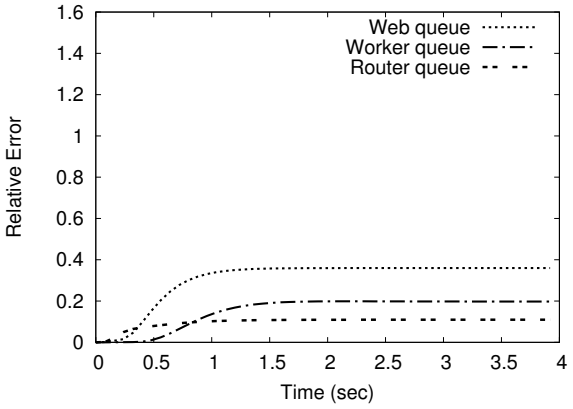


(a) Random routing (QL-based aggregation)

(b) Smart routing (QL-based aggregation)



(c) Random routing (ASE-based aggregation)



(d) Smart routing (ASE-based aggregation)

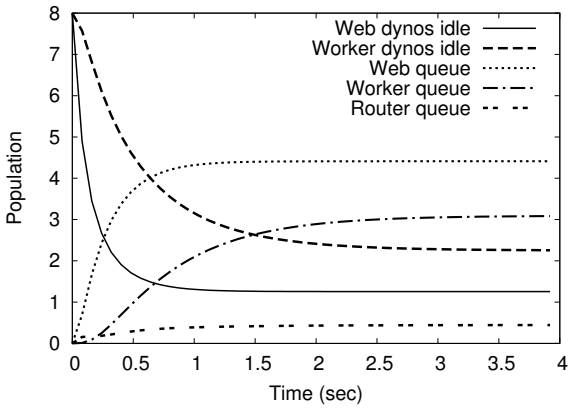
Figure 9: Relative errors in measuring queue lengths for web dynos, worker dynos and the router, given $r_{request} = 40$

between idle dynos and the corresponding average queue lengths has been portrayed accurately enough to show the different behaviour of the two routing policies. Regarding the ASE-based approach in Figs. 10(e) and 10(f), the results are more accurate most of the time. As can be also seen in Fig. 11(c), the errors for the random routing policy are negligible, in contrast with the QL-based method in Fig. 11(a). However, ASE-based aggregation did not produce just as accurate results for the smart routing policy this time, as can be seen in Fig. 11(d). More specifically, it appears that the web dyno queue has been overestimated, while we have accurate estimations for the rest of the queues. Nevertheless, ASE-based aggregation still outperformed the QL-based method, which resulted in greater total error, as seen in Fig. 11(b).

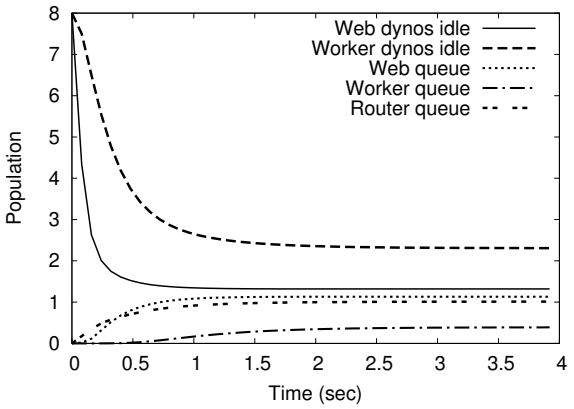
To summarise, the smart routing policy results in better utilisation of the system resources compared to random routing, judging by the number of requests that remain in the queues at the dyno level. Smart routing results in a significantly shorter average queue length, regardless of the workload. Applying compositional aggregation has led us to the same conclusion at a significantly lower cost, using either of the two partitioning approaches considered. However, the approach based on strong equivalence resulted in significantly more accurate approximations for the Heroku example.

8.3. Comparison with Fluid-Flow Analysis

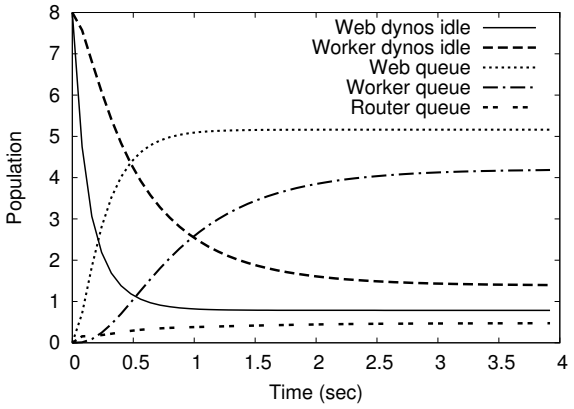
Fluid-flow analysis [20] constitutes a particularly efficient approach to have a deep insight into the system's behaviour not only with respect to the stationary measures, but also the transient ones. In short,



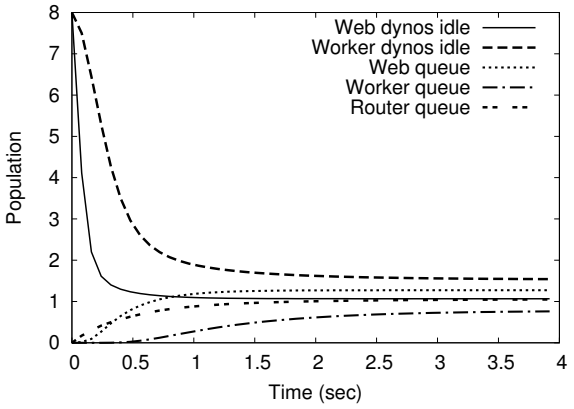
(a) Random routing (Original)



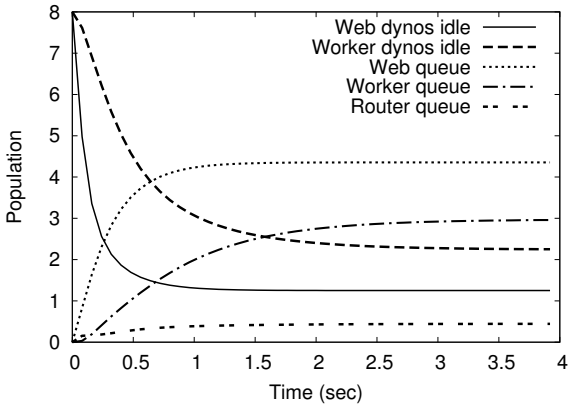
(b) Smart routing (Original)



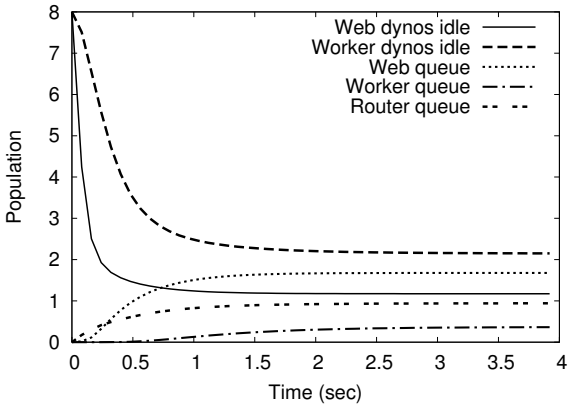
(c) Random routing (QL-based aggregation))



(d) Smart routing (QL-based aggregation)

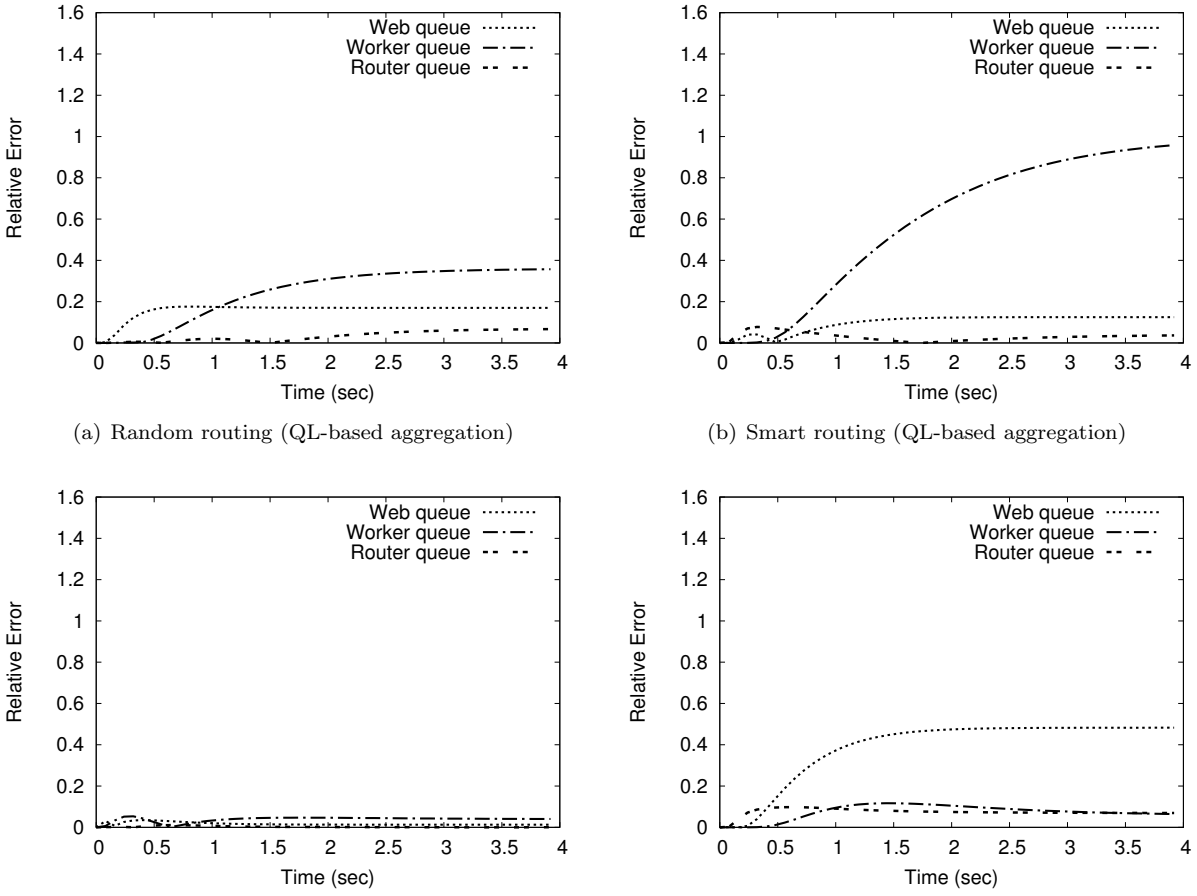


(e) Random routing (ASE-based aggregation)



(f) Smart routing (ASE-based aggregation)

Figure 10: $Random_{8:8}$ and $Smart_{8:8}$ results for $r_{request} = 60$



(a) Random routing (QL-based aggregation)

(b) Smart routing (QL-based aggregation)

(c) Random routing (ASE-based aggregation)

(d) Smart routing (ASE-based aggregation)

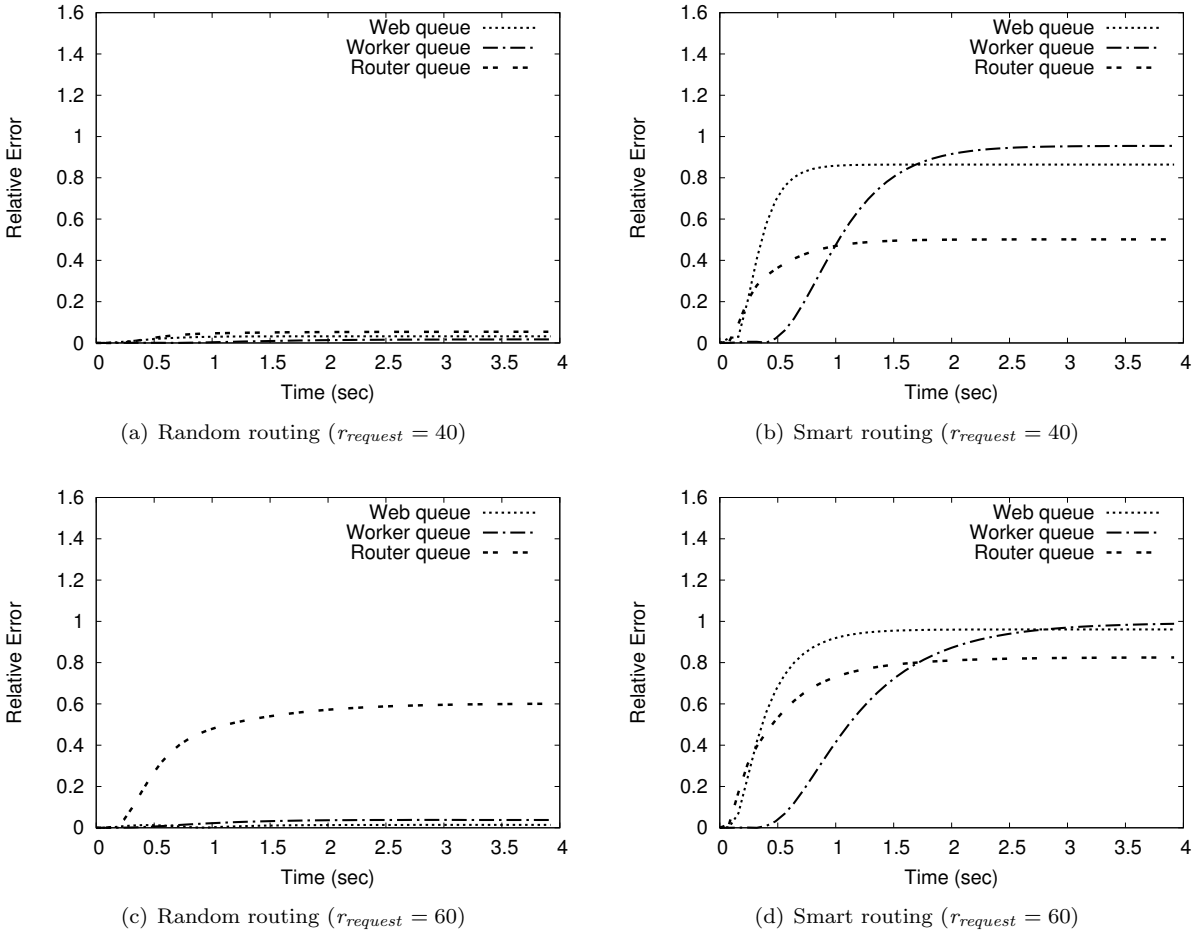
Figure 11: Relative errors in measuring queue lengths for web dynos, worker dynos and the router, given $r_{request} = 60$

the evolution of the population expectations is approximated by a system of *ordinary differential equations* (ODE), meaning that the system is treated as continuous and deterministic. The premise of compositional aggregation has been the efficient analysis of medium-to-large multi-scale systems, for which more efficient approaches such as fluid-flow approximation are not as readily applicable. A multi-scale system is characterised by the presence of both low and high population components, therefore the existence of low population numbers violates the assumption of continuity that the fluid-flow method depends upon.

The Heroku model that is under investigation in this section is a typical example of a multi-scale system; a large number of web and worker dynos that are coordinated by a single routing server. Figure 12 summarizes the relative errors in measuring the queue lengths as given by fluid-flow analysis, for all of the iterations of the routing models considered. We see that the method results in poor approximation in most cases, especially when compared with ASE-based aggregation in Fig. 9 and 11. This result is in line with our theoretical expectation that compositional aggregation can be a viable alternative to fluid-flow when the continuity assumption is violated, as in the case of multi-scale systems.

8.4. Aggregation vs Approximation Quality

By imposing a certain compression ratio, we essentially decide the size of the aggregated component, and subsequently the size of the entire aggregated model. By the term "compression ratio", we refer to the ratio of the size of an aggregated component to its original size. In this experiment, we explore how



(a) Random routing ($r_{request} = 40$)

(b) Smart routing ($r_{request} = 40$)

(c) Random routing ($r_{request} = 60$)

(d) Smart routing ($r_{request} = 60$)

Figure 12: Relative errors in measuring queue lengths for web dynos, worker dynos and the router for the fluid-flow method

approximation quality is affected by different compression ratios for the two aggregation schemes discussed in this work. Moreover, we examine whether the effect of a particular compression ratio is different when aggregation is applied to systems of different size.

We consider the model of the smart routing policy with $r_{request} = 60$. This time, we look into the relative errors regarding the average component populations at steady-state. The relative error is calculated as in (38), where t is such that the system is in steady-state. The values that we report are the averages of the relative errors for all the PEPA components involved in three different iterations of the smart model: $Smart_{6:6}$, $Smart_{6:8}$ and $Smart_{8:6}$. The experimentation results are outlined in Table 4. In the first column, we can see the compression ratios used for component aggregation. The second column shows the total state-space size of the aggregated Markov chain. The components that have been aggregated are *WebDyno* and *WorkerDyno* with a ratio that varies from 0.5 to 0.7. For each aggregation case, we report the average relative errors for the QL-based and the ASE-based approach. In the general case, one would expect that the more the size of the model is reduced, the less accurate the results will be. For the models considered, we can see that we have lower error values as we increase the size of the aggregated model, regardless of the aggregation approach used in each case. Regarding the comparative performance of the two aggregation schemes, we see that ASE-based aggregation produces higher error values than the QL-based approach when the compression ratio is fixed to 0.5. We have to note however that for the particular compression ratio both methods exhibit poor performance. As the compression ratio is increased, the ASE-based method produces

consistently lower errors.

Table 4: Experimentation with different compression ratios

(a) $Smart_{6:6}$

Component Compression Ratio[a]	Number of States[b]	Average Relative Error at Steady-State	
		QL-based Aggregation	ASE-based Aggregation
0.5	173888	0.466	0.524
0.6	251000	0.364	0.294
0.7	341052	0.287	0.194

[a] Aggregation has been applied to *WebDyno*[6] and *WorkerDyno*[6] independently
[b] The original model has 682276 states

(b) $Smart_{6:8}$

Component Compression Ratio[c]	Number of States[d]	Average Relative Error at Steady-State	
		QL-based Aggregation	ASE-based Aggregation
0.5	411312	0.345	0.368
0.6	593364	0.436	0.210
0.7	807508	0.226	0.225

[c] Aggregation has been applied to *WebDyno*[6] and *WorkerDyno*[8] independently
[d] The original model has 1620612 states

(c) $Smart_{8:6}$

Component Compression Ratio[e]	Number of States[f]	Average Relative Error at Steady-State	
		QL-based Aggregation	ASE-based Aggregation
0.5	410176	0.406	0.455
0.6	593000	0.291	0.218
0.7	803160	0.269	0.227

[e] Aggregation has been applied to *WebDyno*[8] and *WorkerDyno*[6] independently
[f] The original model has 1620612 states

As a final remark, it appears there is a trade-off between approximation quality and state-space reduction. Nevertheless, this result should not be generalised for arbitrary models. Sometimes it should be possible to achieve a state-space reduction that is more accurate than other configurations that involve more states. A characteristic example is the case where the model in question is lumpable; there would be no approximation error, as the aggregation would be exact.

9. Related Work on Approximate Markov Chain Aggregation

The notion of *Near-Complete Decomposability* (NCD) [21] has been used in many works as a criterion to aggregate a Markov chain [22, 23]. Methodologies that rely on NCD typically exploit the eigen-structure of transition probability matrices. The eigenvectors that correspond to the largest eigenvalues of a stochastic matrix are known to convey information regarding strongly connected classes of states in a Markov chain. For example in [24], it is shown that partitioning the state-space depending on the sign-structure of the top eigenvectors will minimise the probability of transitioning between classes. In a more recent work [23, 25], a recursive bi-partitioning approach has been presented which is based on information theory. These techniques require that the Markov chain in question is reversible, as they exploit the symmetry imposed by the detailed balance equation. The identification of nearly-completely decomposable partitions has been extended to non-reversible models in a number of works [26, 22, 27], which make use of appropriate reversible models to approximate a non-reversible Markov chain.

NCD can be thought of as a special case of quasi-lumpability, since for all the states that belong to the same class, the probability of remaining in the class approaches 1, while the probabilities of transitioning to other classes approach 0. The contrary does not hold however; which is that a quasi-lumpable (or even lumpable) model does not have necessarily to be nearly-completely decomposable. The relationship between

those two notions of approximate state equivalence has been investigated in [3]. It is our opinion that quasi-lumpability covers a wider range of state equivalences, and therefore it should be more appropriate as a criterion to aggregate a Markov chain, whose properties are otherwise unknown. Apart from the QL-based method in [3], the most relevant approach in the literature has appeared in [28], where an abstract framework for this kind of optimisation is established. An algorithmic identification of lumpability is examined in [27], however aggregation of non-lumpable partitions has not been discussed.

Finally, in this work we have shown that a compositional treatment of aggregation requires a notion of approximate equivalence that is more informative than quasi-lumpability. In terms of PEPA models, we have seen that approximate strong equivalence captures approximate behavioural equivalence for components regarding the total extent of their activities.

10. Conclusions and Future Work

We proposed a framework for approximate compositional aggregation of PEPA models which builds on our earlier work in [3]. This involves defining component partitioning as an optimisation process, where an algorithm blindly searches for approximate state equivalences over the state-space of components. The ASE-based partitioning strategy that we propose involves the use of a clustering algorithm that minimises an upper bound that implies approximate strong equivalence. Moreover, we have discussed how components should be aggregated in the case of a partition that induces an approximately strongly equivalent aggregated component, and we have explored the effect of aggregation on the structured operational semantics of the PEPA language.

The ASE-based method utilises all of the activities that define the behaviour of a PEPA component, in contrast with the older QL-based approach for component aggregation [3], where the shared activities had been ignored in terms of the partitioning process. This exclusion of the shared activities imposed a limitation regarding the applicability of the QL-method: the behaviour of a candidate component should be dominated by its individual activities. Nevertheless, there is no such requirement in terms of ASE-based aggregation, a fact that renders the method applicable to a wider range of models.

Both partitioning approaches have been experimentally evaluated over a realistic case study. Regarding approximation quality, the ASE-based partitioning approach has produced significantly more accurate results compared to QL-based aggregation [3], which is an outcome compatible with our expectations. The ASE-based method simply relies on more data, and in the extreme case, it may capture exactly a partition that induces a strongly equivalent component, while this is not the case for the QL-based method.

Regarding ASE-based partitioning, we have to emphasise that there is no guarantee that the best possible partition is obtained. As a matter of fact, the method is sub-optimal, even if we assume an optimal clustering algorithm, as it minimises only an upper bound for approximate strong equivalence. In terms of future work, we think that approximation quality can be further improved by the refinement of our partitioning approach so either another tighter bound or the ASE-based measure is directly minimised. Another interesting direction for future work is to derive bounds for the approximation errors, for example in the style of [12].

Acknowledgements

The authors would like to thank Vashti Galpin, who recommended the real-world example used in the case study of Section 8. This work is partially supported by the EU project QUANTICOL, 600708. This work is partially supported by the BBSRC SysMIC grant, BB/I014713/1.

References

[1] J. Hillston, A Compositional Approach to Performance Modelling, Cambridge University Press, 1996.
[2] P. Buchholz, Exact and ordinary lumpability in finite Markov chains, Journal of Applied Probability 31 (1) (1994) 59–75.
[3] D. Milios, S. Gilmore, Compositional approximate Markov chain aggregation for PEPA models, in: Computer Performance Engineering (EPEW'12), LNCS 7587, Springer, 2013, pp. 96–110.

[4] J. Hillston, L. Kloul, An efficient Kronecker representation for PEPA models, in: Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification, Springer-Verlag, 2001, pp. 120–135.
[5] G. Plotkin, A Structured Approach to Operational Semantics, Tech. rep., Computer Science Department, Aarhus University (1981).
[6] J. Kemeny, J. Snell, Finite Markov Chains, Springer, 1976.
[7] S. Gilmore, J. Hillston, M. Ribaudo, An efficient algorithm for aggregating PEPA models, IEEE Transactions on Software Engineering 27 (5) (2001) 449–464.
[8] G. Franceschinis, R. Muntz, Bounds for quasi-lumpable Markov chains, Performance Evaluation 20 (1-3) (1994) 223–243.
[9] G. Franceschinis, R. Muntz, Computing bounds for the performance indices of quasi-lumpable stochastic well-formed nets, IEEE Transactions on Software Engineering 20 (7) (1994) 516–525.
[10] A. Bušić, J. Fourneau, Bounds based on lumpable matrices for partially ordered state space, in: ICST Workshop on Tools for Solving Markov Chains, ACM, 2006.
[11] D. Daly, P. Buchholz, W. H. Sanders, Bound-preserving composition for Markov reward models, in: Quantitative Evaluation of Systems, IEEE Computer Society, 2006, pp. 243–252.
[12] M. Smith, Compositional abstractions for long-run properties of stochastic systems, in: Quantitative Evaluation of Systems, IEEE Computer Society, 2011, pp. 223–232.
[13] A. Jensen, Markoff chains as an aid in the study of Markoff processes, Skandinavisk Aktuarietidskrift 36 (1953) 87–91.
[14] J. Hillston, A. Marin, S. Rossi, C. Piazza, Contextual lumpability, in: Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools, ValueTools '13, 2013, pp. 194–203.
[15] A. Ng, M. Jordan, Y. Weiss, On spectral clustering: Analysis and an algorithm, Advances in Neural Information Processing Systems 14 (1) (2001) 849–856.
[16] U. Luxburg, A tutorial on spectral clustering, Statistics and Computing 17 (4) (2007) 395–416.
[17] I. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufmann Publishers, 2005.
[18] M. Smith, Stochastic Abstraction of Programs : Towards Performance-Driven Development, Ph.D. thesis, University of Edinburgh (2010).
[19] M. Kwiatkowska, G. Norman, D. Parker, PRISM: probabilistic model checking for performance and reliability analysis, ACM SIGMETRICS Performance Evaluation Review 36 (4) (2009) 40–45.
[20] J. Hillston, Fluid flow approximation of PEPA models, in: Quantitative Evaluation of Systems, IEEE Computer Society, 2005, pp. 33–42.
[21] P. Courtois, Error analysis in nearly-completely decomposable stochastic systems, Econometrica: Journal of the Econometric Society 43 (4) (1975) 691–709.
[22] T. Runolfsson, Y. Ma, Model reduction of nonreversible Markov chains, in: IEEE Conference on Decision and Control, IEEE, 2008, pp. 3739–3744.
[23] K. Deng, Y. Sun, P. Mehta, S. Meyn, An information-theoretic framework to aggregate a Markov chain, in: American Control Conference, IEEE Press, 2009, pp. 731–736.
[24] P. Deuflhard, W. Huisinga, A. Fischer, C. Schütte, Identification of almost invariant aggregates in reversible nearly uncoupled Markov chains, Linear Algebra and its Applications 315 (1-3) (2000) 39–59.
[25] K. Deng, P. Mehta, S. Meyn, Optimal Kullback-Leibler aggregation via spectral theory of Markov chains, IEEE Transactions on Automatic Control 56 (12) (2011) 2793–2808.
[26] G. Froyland, Statistically optimal almost-invariant sets, Physica D: Nonlinear Phenomena 200 (3-4) (2005) 205–219.
[27] M. Jacobi, A robust spectral method for finding lumpings and meta stable states of non-reversible Markov chains, Electronic Transactions on Numerical Analysis 37 (1) (2010) 296–306.
[28] S. Balsamo, G. Dei Rossi, A. Marin, Cooperating stochastic automata: approximate lumping a reversed process, in: International Symposium on Computer and Information Sciences, Springer London, 2013, pp. 131–141.